A gentle introduction to SAGA GIS

Víctor Olaya
Edition 1.1 — Rev. December 9, 2004

A gentle introduction to SAGA GIS.

Edition 1.1
Rev. December 9, 2004
Revised by Javier Pineda (AESIA Desarrollo y Proyectos Medioambientales, S.L., Madrid, Spain.) and Victor Olaya.

For any further information, please contact the author at the following e–mail address

volaya@ya.com

This book can be downloaded in PDF and LaTeXformats from the following website:

http://geosun1.uni-geog.gwdg.de/saga/html/index.php

Acknowledgments

This book will not be the same without the contribution of the following people, to whom I am in debt:

- **Javier Pineda** from Aesia Desarrollo y Proyectos Medioambientales S.L, who painstakingly read the text and corrected many of my mistakes. However, if you still find any mistake, I'm the one to blame, not him...
- **Andre Ringeler** from the SAGA Development Team, who supported the manual from the very beginning and showed great interest. Also, the chapter about programming is based in his great text *First steps in SAGA modules programming*, from which I borrowed many ideas and even some text fragments.

Contents

Preface

SAGA is a free Geographical Information System (GIS) with support for vector and — specially — raster data. This book describes the main features of SAGA, and its aimed at every user who wants to use the program whether as a learning tool (SAGA is a very good choice to discover GIS fundamentals) or as an all–purpose GIS with which develop "real" geographical analysis.

No GIS knowledge is assumed, so this work can be used as a self–study reference by all those who want to initiate themselves in the study of Geographical Information Systems.

The book is conceptually divided in two parts: the first one (chapters from 1 to 5) deals with basic SAGA functionalities, while the second one explain how to perform data analysis using some of the most important SAGA modules. Both parts seem rather independent, but they are closely related. You can read the first part and you will get a good knowledge of SAGA and how it works, but if you don't get into the second part you will not be able to get many results. On the other hand, you cannot study modules described in the second part without reading the first one, since they all make use of SAGA core capabilities, which are explained in the first five chapters.

For the technically inclined reader, I have included some information about the algorithms and technical concepts implemented in each module, targeted at those who like to get some beyond nuts–and–bolts descriptions. However, this information is rather limited, since the book will grow overwhelmingly large otherwise. Further references are provided in case you want to get a deeper knowledge of how a particular module works.

The last chapter of the book, which could be considered as a very brief third part, introduces the fundamentals of SAGA module programming. You should only read this chapter once you have become acquainted with the information from the previous chapters. Knowledge of C++ programming is assumed.

If you have ideas for something to be added, removed or altered in this document, please let me know. I would love to get as much feedback as possible from readers.

Chapter 1

Introduction

1.1 Introduction.What is SAGA?

Since you are reading this book, I guess you have some interest in GIS and maybe you already have a solid background on Geographical Information Systems. Perhaps, you have worked with ArcView 3.x (I pity you...) or the more modern ArcGIS (not that bad...), probably the two most popular GIS applications around. Or perhaps you have used GRASS in an UNIX environment and tried its different flavor. Whichever your case, I'm sure SAGA will be be a great complement to all those programs, and you will find it really easy to adapt yourself to it. For someone who has already mastered the basic concept of GIS analysis, SAGA is not just a powerful tool, but also a very enjoyable piece of software.

If you have heard of those applications named before but never used any of them, or if you never heard of them, then SAGA is a great software to get you introduced into the wonderful world of Geographical Information Systems. SAGA is amazingly powerful for raster analysis, while still being very easy to learn. Also, it can handle vector data and supports the most popular file formats for both data types, so you will have no problems when using geographical data from many different sources. I bet that, no matter how complex the problem you need to solve, you will never run short of raster capabilities (maybe the time–series support that can be found in IDRISI or PCRaster is one of the main lacks, thought not a very important one). From the ground up to a more serious usage, SAGA is an interesting option to consider, since it combines a powerful geographical analysis engine with an intuitive and user–friendly GUI.

Most of all, SAGA is free software, so you can use it, distribute it and even modify it freely (have a look at the GPL license under which SAGA is releases if you want more details). Neither ArcGIS, nor IDRISI, nor PCRaster are free. GRASS is free (also released under GPL), but in the time it takes to learn how to use every single module and function in SAGA you would not have done but learn how to install a GRASS distribution (if you can get any further in that time, I guess you must be some sort of GRASS wiz.... Then, you probably won't need this book...)

Last, but not least, you can develop your own SAGA modules in case you need them. Not a lot of programming knowledge is needed, you can make a complete module with 100 or 200 lines of C++ code. If you have a deep knowledge of C++ programming, you can even modify the SAGA API to create a tailored version that will fulfill all of your necessities.

All this sounds really good, but... how can I learn how to use SAGA? At the time of writing this book, there is no SAGA documentation at all, even although the software itself is quite mature (V1.1 was released on 06/03/2004). There's a *Help* menu item, but no help file to go to...

Right now, SAGA is not a widespread GIS (something that I hope this book will help to

change), and I believe that one of the main reasons that avoid SAGA becoming popular is the lack of documentation. So the main aim of this book is to shed a ray of light over all the great functionalities implemented in SAGA, so more and more people can enjoy them and discover a good alternative to mainstream GIS applications.

1.2 Who has created SAGA?

You might not be really interested in knowing who has created SAGA or how the program has evolved since its first versions, but I believe that credit should be given to those who have created such a great GIS software. Also, this book wouldn't exist if SAGA didn't exist!!

In case you like SAGA (and I guess you will), you will know who to thank for it. In case you hate it (not likely...) then you will now who to blame.

At this moment, SAGA is being developed and the Goettingen Univerisity, in Germany, by a small group of developers, but, since SAGA is an Open Source project, developers worldwide can join it and help. Helping does not mean just writing lines of code, but also testing, debugging or doing any other task that can help SAGA become a better software. I like to think that, writing this book, I am contributing to the SAGA project, and I would like to use this lines to invite you to contribute as well if you consider that it is worth the effort.

The roots of SAGA can be found in a program called DiGeM, which was created by Olaf Conrad not so long ago. He took the main algorithms of DiGeM, changed the structure of the program to "expand" it and make a real GIS out of DiGeM, released the source code, and created SAGA. Right now, he works in the development of SAGA, supported by other programmers from the Goettingen University.

To contact SAGA developers, you can write an email to the following email addresses:
Olaf Conrad : oconrad@gwdg.de
Andre Ringeler : aringeld@gwdg.de

1.3 How can I get SAGA?

SAGA can be freely downloaded from the Internet from several websites. Since it is free software, anyone can freely distribute it, whether in its original version or in a modified one. The main place to go is the SAGA official web site located at

http://geosun1.uni-geog.gwdg.de/saga/html/index.php

If this is too long for you to remember, you can also go to the SourceForge SAGA website, located at

http://sourceforge.net/projects/saga-gis/

or, even better, this address will redirect you there:

http::/www.saga-gis.org

From there, you can download both the source code and the compiled binary files for Windows. If you plan to develop your modules or modify the SAGA API(module development is explained in the last chapter of this book, but SAGA API modification is outside its scope), you will need the source code files. Otherwise, simply download the binaries.

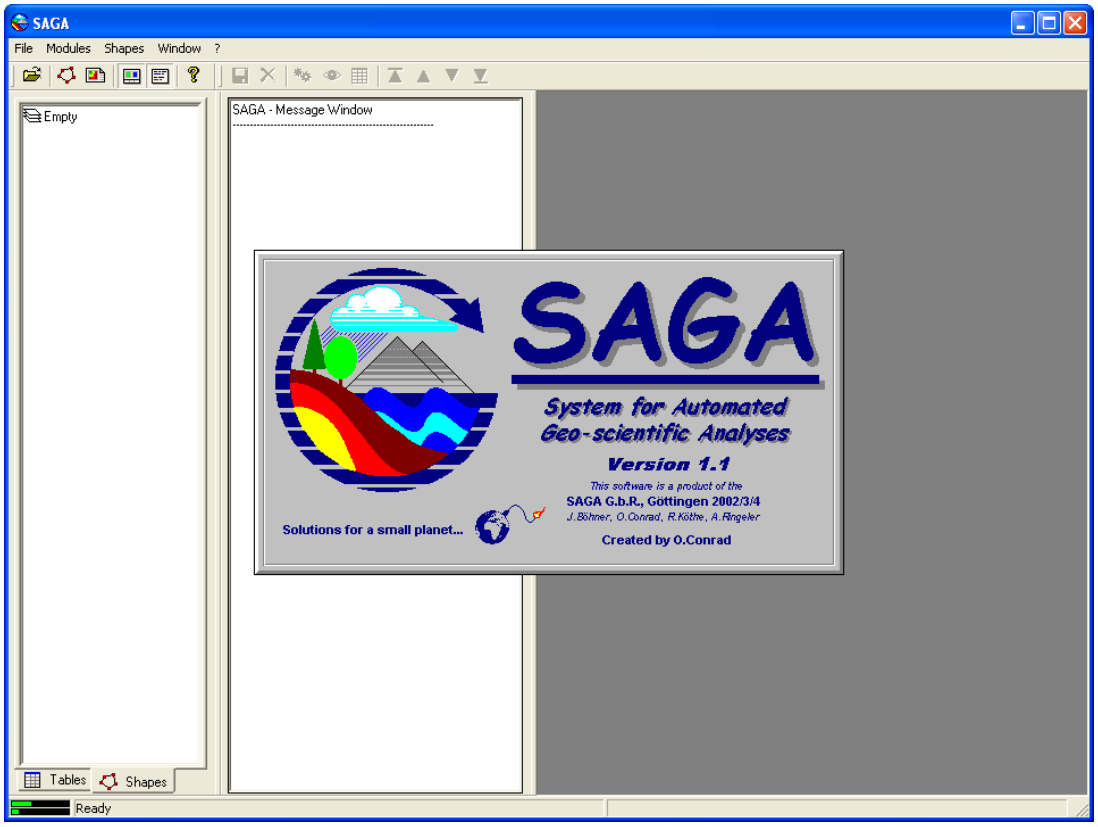
Some "unofficial" modules also have to be downloaded, particularly the ones that I myself have developed. You can find them in the *Modules Contrib* link in the download section of the

SAGA website, and they are identified by my own name. Download them all (only in binary form, not the source code) and extract the zip files into the folder called *modules* under the folder in which you unzipped the main SAGA file.

1.4 Installing and running SAGA

After downloading the saga.zip file, you will have to install it. To install SAGA, just unzip that file in a folder of your choice, and it will automatically create the required file structure. This cannot be really called an installation process, since there is no installation software and it does not modify the windows registry or any file outside SAGA's own folder. That means that you can install SAGA even if you are not the system administrator, and that you can "take" SAGA from one computer to another by simply copying the directory were it is located.

To execute SAGA, open a file explorer, go to the folder where you copied SAGA, get into the *bin* folder and double click on the `SAGA-Lite.exe` file. You will see the following window.



To hide the splash window, just click on it.

A note for Linux users: as you might have noticed, SAGA is a Windows application. I haven't tested it myself, but SAGA has been reported to work well under the WINE emulator. A native Linux version might be on the way...

Chapter 2

SAGA fundamentals

2.1 Introduction

SAGA is now loaded and we can start working with it.

This chapter presents the main features of SAGA and the SAGA GUI. It is aimed at providing the reader with a basic knowledge of how to operate the software, open data files, define projects and perform essential operations with spatial data, and also at describing the structure of the program. Consequently, a fundamental base is laid that will permit the comprehension of the following chapters.

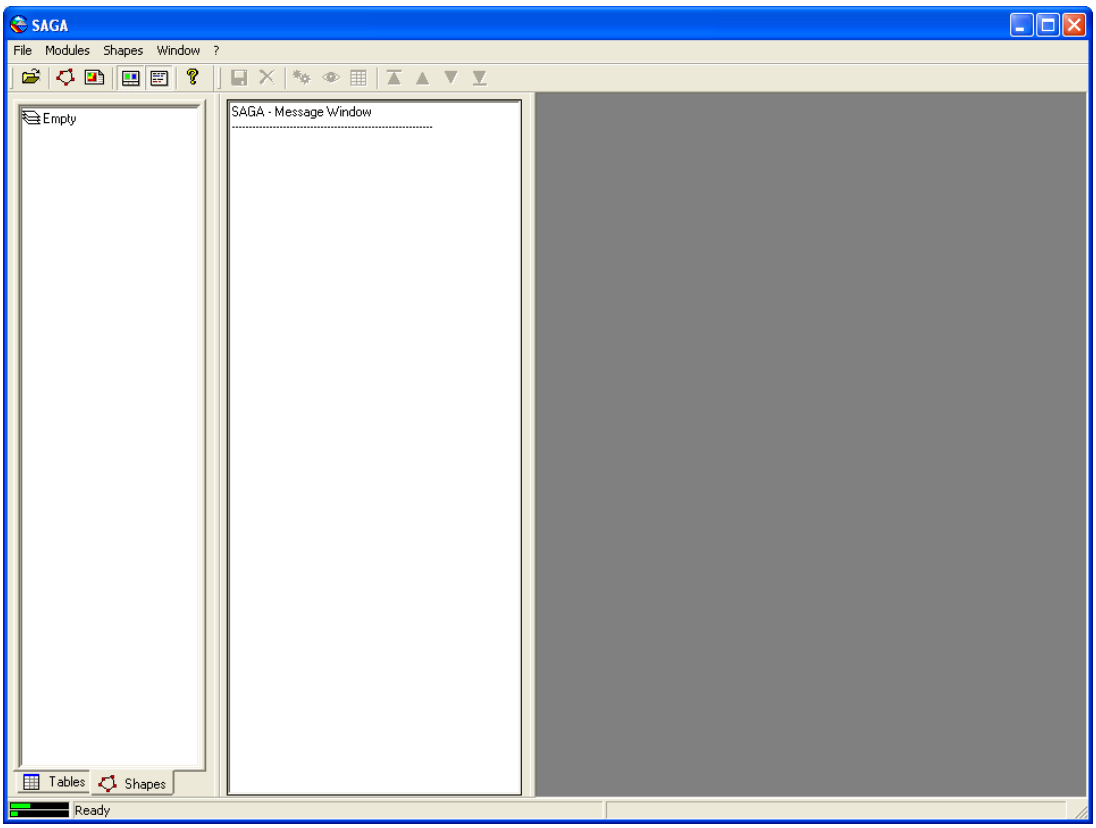
Although, due to its modular structure — more on this will be explained later on —, SAGA's features and capabilities might change depending on which modules are being used, all of the functions explained in this chapter — and also in the next two ones — comprise the core of the SAGA structure and, therefore, are common to all possible "settings" of the software and can be found in every SAGA execution. By mastering all the information included in this chapter, you can fully understand the most important aspects of SAGA and its underlying philosophy, and then extend your knowledge to other more specific aspects.

2.2 The SAGA GUI

The SAGA GUI (Graphical User Interface) represents the linking element between the user and the saga API. The GUI has a pretty simple structure that allows working with many different data sources and results, while keeping all of them correctly organized. Fortunately, it has no complex features, and the most relevant ones require just a few mouse clicks and can be mastered almost without significant effort (relax, no steep learning curves here at the beginning, everything its quite intuitive and easy to learn). It will not take long before you can obtain your first results with SAGA

If you have used any other GIS — specially ArcGIS —, you will easily adapt yourself to the design of the SAGA GUI. Anyway, if that is your case, take your time to read this chapter — skip it only if you feel like you can learn this kind of stuff by yourself by trial and error —, since there are some distinct features — some of them very interesting and useful — that you have probably not seen in other similar GIS apps.

The main SAGA window, when loaded for the very first time, has a look like the one shown below these lines.



Of all the things that can be found in this first window, five of them are worth mentioning, namely

1. Menu bar
2. Status bar
3. Messages window
4. Toolbar
5. Project window

The menu bar is the main element of user interaction, so we will start with it.

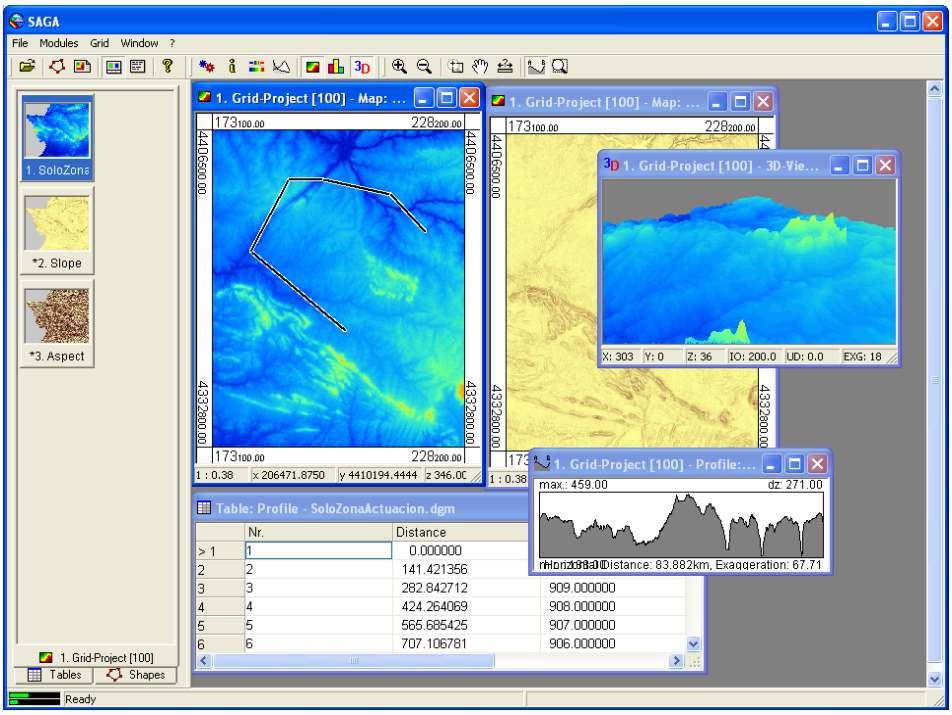
While some menu bar items remain unchanged whatever kind of data you are working with, some others only appear when dealing with a particular element. For example, the *Project* and *File* menus will always be there, since they are related with the project as a whole, but others like the *Profile* menu (not shown in the above image) will be accessible only if a profile is being drawn, something that rather seldom occurs. This context dependency leads us the definition of six main contexts, with which the user should become acquainted.

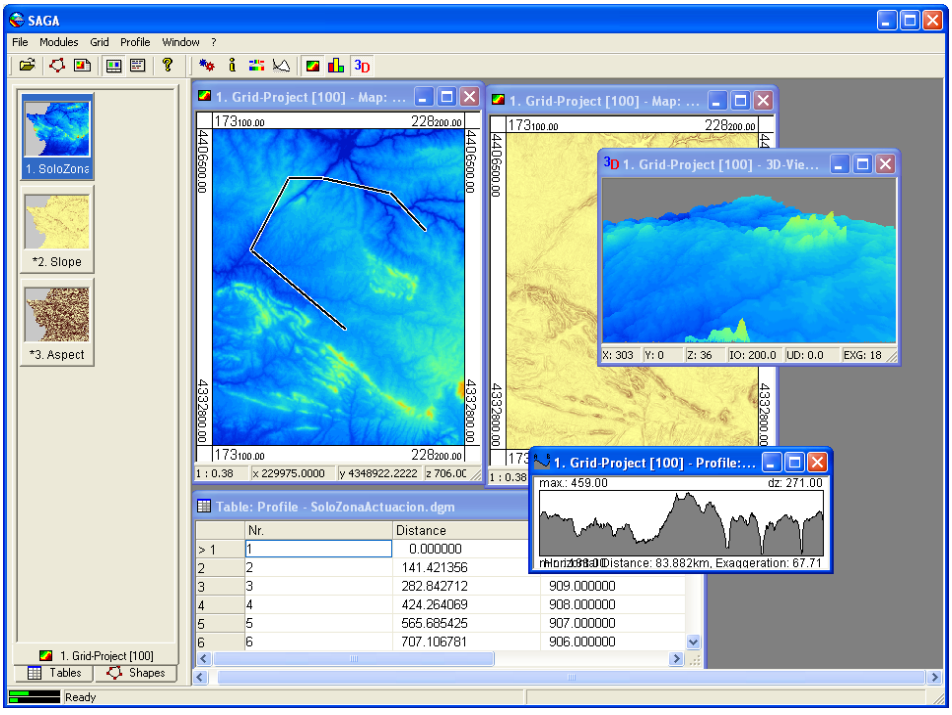
1. Shapes
2. Grids
3. Tables
4. Profiles
5. Layouts

6. 3D–Views

Except for Profiles and 3D–Views (which have been included in the *Working with grids* chapter), they all have a special chapter dedicated to each one of them, and will be later studied in detail. By now, you just have to know that the elements that can be found in the menu bar are not always the same, and you should consider this when you want to use one of the "volatile" ones, so you know exactly in which context you have to look for it.

Have a look at the two following images.





Both of them have plenty of windows opened (pretty impressive, isn't it? Don't worry about how they were all put there. You will soon learn to open data files). Although they

look almost the same, a closer look reveals that in the first one the window containing a grid is activated (see the different color?), while on the second one the profile window is the active one. If you look at the menus, they are not the same. That means that the context is defined by the data type contained in the active window at each time.

Since the toolbar presents some of the most important and frequently used menu items in a more handy way, it is also context–dependent and will change as you change from one data type to another. By placing the mouse pointer over a button in the toolbar, the program will show a little label with its description. Therefore, they are not analyzed here, since the information about their behavior can be found in the corresponding menu item that they represent.

Let's go now to the project window. This window contains all the data that has been loaded into SAGA (or generated by any of its functions and modules), which is divided into three groups.

1. Shapes
2. Grids
3. Tables

Even when no tables or shapes are loaded, a corresponding tab will be shown in the lower part of the project window. In this case, an *empty* label will be put in the window itself. On the other hand, if no grids have been loaded (or generated), no grid tab will exist. That is why now, at the beginning, you can only see two tabs in the project window. The main reason for this is that grid projects are divided according to their resolution (the cell size of the grids they contain), and SAGA can handle not just one project but many of them. While only one shape or table group is needed, the number of grid groups (and tabs) needed depends on the loaded grids.

Regarding the message windows, it is shown by default by SAGA, but it takes a large space of the screen. Most users find it better to hide it and only show it when there is a problem with the execution of a module or any other component of SAGA, so they can trace the source of the error and correct it. To do this, use the menu item *Window/Messages/Show*, which should have a tick. Click on it and the tick will disappear, as will do the messages window.

The project window can also be hidden by using the menu item *Window/Project*, but it is not likely you will want to hide it.

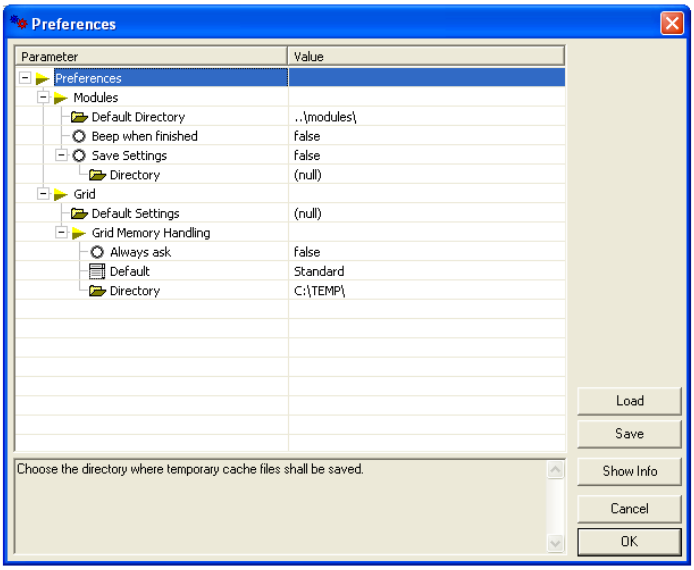
Both the project window and the messages window can be resized by placing the mouse pointer over the edge of the window and clicking and dragging to expand or shrink it as desired.

Finally, the status bar displays information about the activity of the program. On the right side there is a progress bar (right now you can't see it, for there's no activity going on), and on the middle you can find textual information (right now, the program should display a *Ready* label, indicating that it is prepared to perform any required operation).

2.3 Preferences. The parameters window.

Once the main components of the SAGA GUI have been described, and before we start incorporating spatial data into the program, it's time to configure it so the next time we start SAGA it is adjusted to our particular needs. Not a lot of parameters can be adjusted, but we will use this brief explanation about the configuration of the program to introduce the main component that SAGA presents when it needs an input from the user. This is what we will call the *parameters window*

To configure SAGA, go to *Preferences* under the *File* menu. You will see a window like the one below.



Whenever SAGA requires an input from the user, it will present a window with the above structure. You will find a two column table in which the left column includes a list of parameters and the left one the values of those parameters. As expected, the user cannot interact with the left part, but can modify the values found on the right one.

Some parameters have a description that might help the user understand their meaning. When you click on a parameter, its description, if it exists, will be shown in the lower part of the window, in the gray box. You can hide this box by clicking on the *Hide Info* button.

Many different kinds of parameters can be presented using this scheme, and we will study them in detail as they appear. Right now, we will focus just on the ones that appear on the preferences window.

First, we will set the parameters related with the usage of modules. Most of the SAGA functionalities are included in modules which are themselves grouped in libraries. As we will see when we reach the corresponding section, those libraries have to be loaded into SAGA in order to be able to access their modules and use them. Since those modules are so important and are constantly used within a typical SAGA session, it is a good idea to load them automatically when SAGA is started. To do that, the path to the folder where the main module libraries are stored must be introduced in the *Default Directory* cell. Click on this cell (of course, on the *Values* column) and a dialog box will pop up. Select the aforementioned path and then press *OK*.

If you remember from the *Installing SAGA* section, modules are kept in a folder called *modules* (quite logical...) under the SAGA installation folder.

As you can see, the parameters cannot only be set using the keyboard and directly writing its values. Some cells, when clicked, trigger different actions and show dialog boxes that can be used to introduce the information required in a more practical manner.

The two parameters below the *Default directory* node are boolean parameters (that is, they can be true or false). If you click on them, a list containing true and false will show. Both parameters, an also the remaining one under the *Modules* node, are self–explanatory, so you should be able to figure out their meaning. If not, have a look at the lower info box and read. Configure them to your personal preferences (personally, I find the beep option rather annoying...).

A very important feature regarding parameters in SAGA is that you can save the values of these parameters for a later use. In this case, there are not a lot of parameters, and they can be adjusted in seconds, but some other functions require a greater amount of information. For example, the parameters that control how grids are displayed are very numerous. In SAGA, you can set these parameters only once, and let the program load them each time it is started. That doesn't mean that you cannot change them again, but it means that, by default, the way grids are displayed is exactly how you like it.

To do it, click on the *Default Settings* line and a new dialog box will appear. You will be asked for a Saga Parameters File (with the spf extension). Since we do not have one (and will not have one related to grid displaying until we reach the *Working with grids* chapter) just close the dialog clicking on the *Cancel* button. We will see how to load and save parameters files.

Grid displaying parameters are the only ones that can be set by default, but almost every single function in SAGA require some parameters to be adjusted. Therefore, it is really useful to save some predefined configuration to save time when working with large sets of these parameters.

Once you have adjusted a set of parameters, you can save it by using the *Save* button. Whenever you want to recover the recorded values, just use the *Load* button and select the corresponding parameters file in the dialog box that will appear.

If you mix up different parameters file (that is, for example, if you save the parameters in the preferences window and later on load them when adjusting the parameters that control another area of the program), since they refer to different parameters, it will have no effect at all.

Going back to the SAGA preferences window, the parameters under the *Grid Memory Handling* node control how grids are stored in memory. They are a bit too complex to be explained now so, by the moment, leave them unchanged.

Click on the *OK* button to close the window.

2.4 Projects. File handling

It is time now to load our first spatial data and start discovering how SAGA can handle it and generate results from it.

As we have seen before, data in SAGA is organized according to its nature. For that reason, there are tabs in the project window and different contexts (as we have called them) exist within the program itself. However, spatial data can be divided in two main groups, each one of them featuring its particular functions and command in SAGA. These two are the well–known raster and vector groups, from which the user has probably heard before.

It's not the aim of this book to present the theoretical differences between raster and vector data or analyze the suitability of each one of them for a particular task. I recommend having a look at the references and getting a good book about the subject (there are plenty of them). Although it is not necessary to have a deep knowledge about raster/vector fundamentals, the user should become acquainted with those concepts before starting to work with spatial data.

Assuming that you have some sort of notion about the differences between raster and vector data, we will try to outline how both data types can be incorporated into SAGA and how they are organized.

Although some vector capabilities are included in SAGA (and they are increasing in each new release), SAGA still remains a raster GIS, mainly due to the great number of raster analysis modules in comparison with the rather limited amount of modules that perform their operations on vector data.

In the course of a SAGA session, many raster or vector files can be opened, each one of them containing information about a particular parameter of a geographical area. All this information can be grouped into a project, so the next time that this area has to be studied all the files can be loaded at once instead of file by file.

In the following pages, we will study how to open data files and how to create and use projects, whether they are comprised of raster layers or vector layers (the term layer will be used indistinctly for raster and vector data, and you will find it in almost all the GIS–related texts).

First of all, let's open a grid file with a Digital Elevation Model. From the same website where you have downloaded SAGA (and maybe this manual), you can also get some demo data. Download the demo.zip file and unzip it wherever you want.

Now, select the *Open...* menu item in the *File* menu. In the dialog that will appear you can select the file you want to open. From this dialog you can open raster and vector data, and even tabular data. However, only a few file formats are supported, and the raster ones are not very usual and not supported by other GISs. You will not have any problem with vector data, since the ESRI shapefile format (the most popular one for vector data) is supported, but you will not be able to open raster layers in some of the most popular file formats. To open those files, Import/Exports modules have to be loaded. This subject will be explained a couple of chapters later. Right now, we will just focus on the file formats that are supported by SAGA without needing any additional modules.

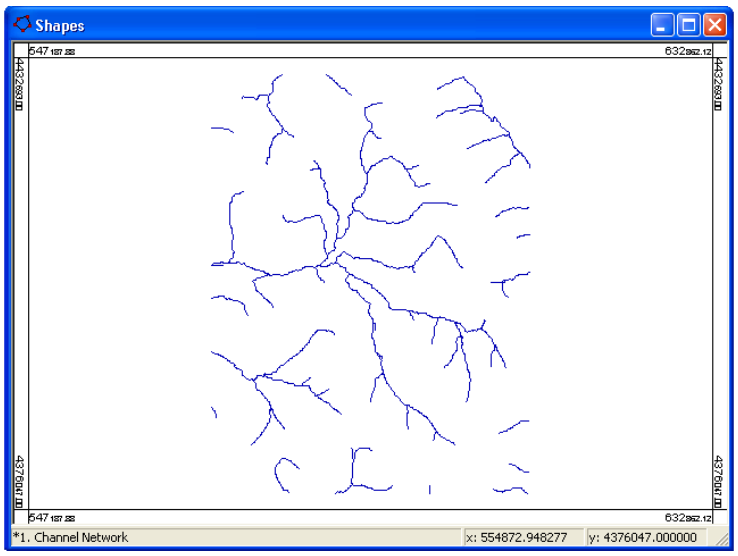
The "native" saga format for raster data is the DiGeM file format. DiGeM is SAGA's predecessor and was developed by the same author of SAGA. Consequently, the author has kept the implementation of this file format from one program to another.

In the folder where you have unziped the demo.zip file, select the file named test.dgm. You can select more than one file at a time, by using the shift and control keys as with any other windows application. When using Import/Export modules, you will only be able to open files one by one.

If you have already opened the test.dgm file, the first thing you will notice is the new tab in the project window. It should say "1. Grid-Project [90]". That "90" means that the cell resolution equals 90 meters.

From now on, if you open (or generate) any other grid file that represents the same geographical area and has the same cell size as the already loaded one, it will be included into this project. If it has a different cell size or represent a different geographical area, a new project will be created automatically. For example, if you open a raster layer with a cell size of 200 meters, a new tab will appear with the label "2. Grid-Project [200]".

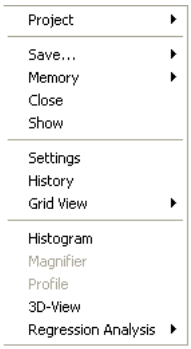
Let's go now for a vector file. As before, select the *Open...* menu and open this time the file called test.shp. A new window will appear, showing some lines that represent a channel network.



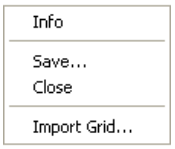
Although this channel network corresponds to the same geographical area as the raster layer already opened (in fact, it has been created using it), they are kept in isolated windows. SAGA separates raster and vector data (as opposed to, for example, ArcGIS), but we will soon see that they can be combined without much effort, at least for visualization purposes. When dealing with raster data, a vector layer can be added, but the raster settings will rule the display and the menus for raster layers will be shown in the menu bar. The same will happen when working with vector data.

You can alternate between the raster and vector tabs in the project window. Raster layers are represented using a small thumbnail, while vector layers are presented simply by their name.

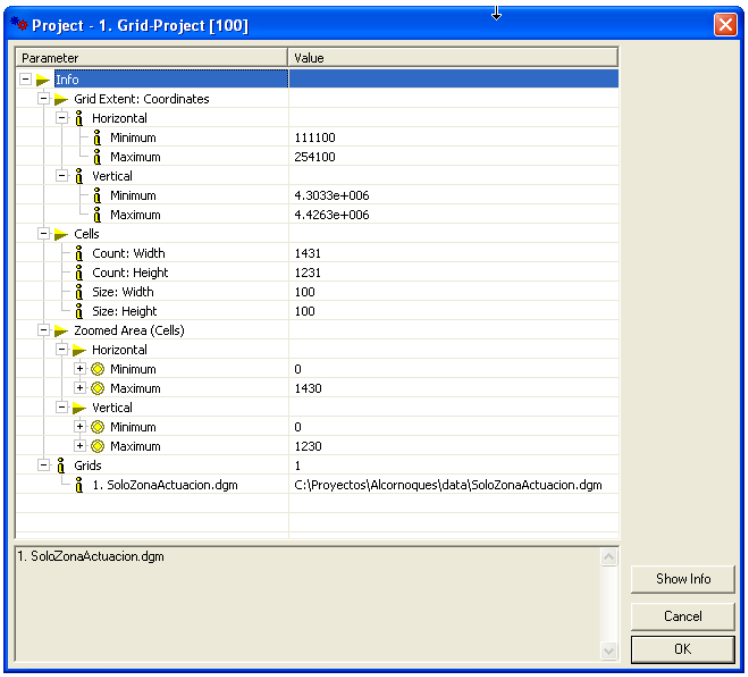
Select the Grid–Project tab and then right–click on the project window. You will see the following pop up menu



If you place the mouse cursor over the project menu item, a new menu will get unfolded.



Select the *Info* menu item and you will get information about the grid project and the files that are included in it.

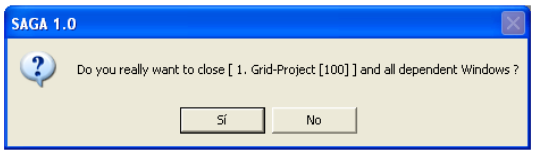


This time, you cannot edit the information in this window. This is not a data input window, but an informative one. Can you see the *Load* and *Save* buttons for loading and saving parameter sets? Of course not! They would not be especially useful here...

Close the window by clicking on the close button.

From the previous menu you can also select the *Project/Save...* menu item. It allows you to save the structure of the actual project, so you can open it later with just a single mouse click. To experience how projects work in SAGA, follow this steps.

- Select the Save menu and in the dialog box that will appear type a name for the project (i.e. testproj.spg). Click the *OK* button. The project is now saved.
- Close the grid project by selecting the *Project/Close* menu item. SAGA will ask you if you really want to close it. Press the *Yes* button.



  Right now, there's no grid project and, therefore, no raster data tab in the projects window.
- Re–open the project you have just saved (and closed), by selecting the *Open...* menu item under the *File* menu and then selecting the filename you entered when saving the project.

As you can see, the *File/Open...* menu item can be used to open all kinds of files, whether single data files (vector and raster) or whole projects (also vector and raster).

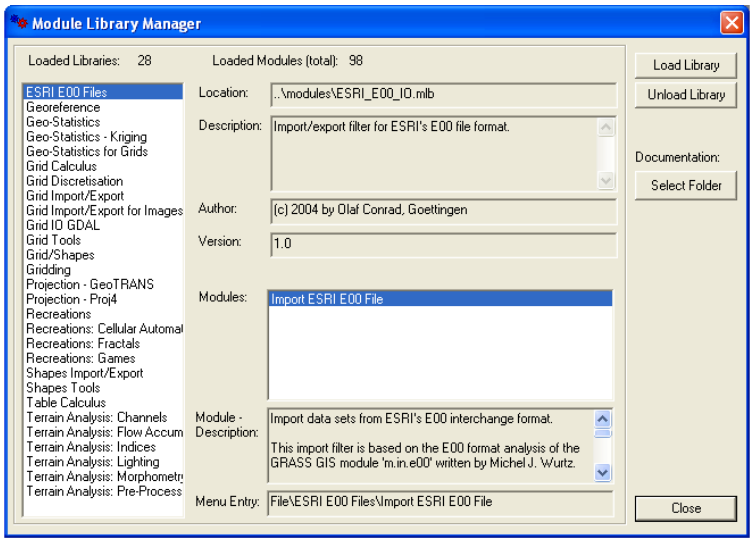
If you want to work with a vector project, just select the *Shapes* tab in the project window and then right click in the window itself. A pop up menu will appear, which contains similar menu items as in the raster case.

2.5 The SAGA module structure

As it has already been said before, most of the functionalities of SAGA (among them, the more interesting and powerful ones) are included in module libraries that have to be loaded before using them. Some of those modules are provided with SAGA itself and included in the SAGA distribution, while others have been developed by external collaborators (or you can even develop your own ones if you feel capable...)

We already saw in the *Preferences* section how to load some modules by default. However, only a folder can be specified as the default for modules, and you might have several modules in other different folders. If you want to use these modules, you will have to make use of the Module Library Manager.

The Module Library Manager can be called by selecting the *Module Library Manager...* menu item under the *Modules* menu. A window like the one below will appear.



Using the Module Library Manager you can load module libraries into memory, but also unload them in case you need it. If some libraries are already loaded (and they will be if you have correctly set the default path for modules before starting SAGA), you will see their names in the left side of the manager window. If you click on any of these names, the modules included in the selected library will appear in the box labeled *Modules*. Clicking on any of them will cause its description to appear in the text box below.

To load a library, click on the *Load Library* button and then select the library you want to load. Its name will appear along with the already loaded ones.

To unload a library, select it in the left list and then click the *Unload Library* button.

When developers create their libraries, they usually add some information about them in the source code, describing the parameters that must be supplied to each module or including some references where further information can be found about the implemented algorithms. Some of this information will appear in the info box that we have already seen as a part of the parameters window. This "inner" documentation can be "taken out" of each library and each module, and formatted in order to create a good documentation resource. SAGA itself is capable of automatically create documentation in HTML format which can be later used to know the characteristics of each module.

You can control where in your hard disk this HTML files are stored, by specifying a folder using the *Select Folder* button.

To read the generated documentation, from the main menu (outside the Module Library Manager), select the *Module Libraries* menu item under the *?* menu. Once the HTML files

have been created, you can also view them using you favorite web browser.

To close the Module Library Manager, use the *Close* button.

If you just want to load a single library (or a bunch of them), and you have no need of knowing which modules are included in each one, or you don't want to unload any other library, using the Module Library Manager might be a bit awkward. In this case, you can use the *Load Library* menu under the *Modules* menu.

Whichever method you use to load you libraries (or even if they are all loaded at the beginning of the SAGA session), every module library will have its menu item and all its modules will have their one under the latter. That means that, as you load more modules, the *Modules* menu under which all of them can be accessed will get populated. That also means that most of the usual operations that can be performed using SAGA are found under the *Modules* menu, and, consequently, it should be available whatever the context you are in. Since the module structure of SAGA allows the development of modules that deal with raster data, vector data, both of them or none of them (in fact, you can even develop modules that don't perform any spatial analysis or have any GIS component...), access to modules should not be context–dependent.

However, modules itself can be context dependent. If no raster layer has been loaded an a module takes raster data as input, you will probably not be able to use it.

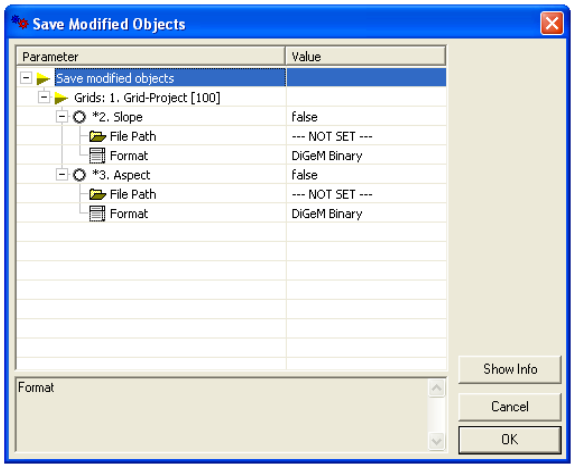
Several chapters are dedicated in this book to some of the most important modules, describing their particularities. However, right now you just should learn that modules gather under the *Modules* menu, and that calling them is as simple as clicking in their menu item.

2.6 Closing SAGA

After finishing a SAGA session, just click on the upper right corner of the main window or select the *Exit* menu item in the *File* menu.

If you have opened any grid or shape using the Import/Export modules (that is, if you have opened any "non native" file), SAGA will ask you to save those files. Also if you have generated any grid or shape layer and not saved it, it will also prompt you, in order to avoid data losses.

You will see a window like the one below.



If you want to save any (or all) of the layers you have been working with, just select the desired format and specify the path to which it must be saved. After setting these parameters, just press *OK* and SAGA will be closed.

Chapter 3

Working with grids

3.1 Introduction

You should already know how to load raster and vector layers (at least in those formats supported by SAGA without the need of any additional module), and create projects with them. It's time now to start working with those layers and discover how SAGA can handle them and offer some nice results.

Although the most "intensive" procedures and algorithms are kept in modules, SAGA itself includes a nice set of functions that prove themselves really useful for the everyday GIS work, and a great amount of them are targeted to grid analysis. For this reason, and also because SAGA is mainly a raster analysis tool (a nature inherited from its predecessor DiGeM), we will deal with grid functions now and leave vector stuff for a later chapter.

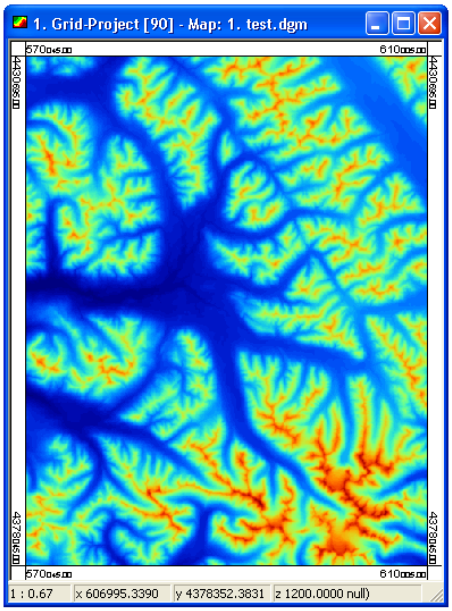
However, raster layers are not only the source data for performing analysis, but also something with which the user can interact, and something from which he can obtain other results, particularly those related to its visualization and displaying. Therefore, before beginning to extract information from raster layers we will have a look at how SAGA displays grid layers and how the user can perform this interaction.

Also, some Input/Output functions (specially output) will be analyzed, completing what was previously said about the handling of raster data files.

3.2 Displaying grids. Basic tools

The first thing to do to display a grid is simply double–clicking on its thumbnail[1]. If you loaded the test.dgm file, you will get a window like the one shown below.

[1]Linux users might have problems with this. Open the grid view window using the popup menu instead



In it, you can see the Digital Elevation Model with a great level of detail, identifying high areas (depicted with brown and white tones) and low areas (with green and blue tones). The window in which the grid is displayed has a status bar in it with four little text boxes. From left to right, these boxes contain the following information.

- **Scale of the representation**: this is NOT a cartographic scale but rather a "pixel" scale. It shows the ratio between the dimensions of the grid (in cells) and the dimensions of its screen rendering (in pixels).
- **X coordinate of the mouse pointer**: If you move the mouse over the image, you will see how this value (and also the following two ones) change. Also, if more than one grid window is opened, although you move the mouse only over one of them, the values in the status bars of the other windows will change as well.
- **Y coordinate**.
- **Z value**: The value of the cell the mouse is over. In this case, since it is a DEM, this value represents the elevation of the cell.

Surrounding the grid you can see the coordinates of its four corners.

Now, let's focus on the lower areas of the terrain (the ones with a lower height), which are located approximately in the center–left part of the grid. SAGA has the usual zoom functions that allow the user to reach the desired level of detail and closeness (of course, always within the limits imposed by grid resolution), or restrict the display to a particular area.

Four main buttons can be used to select four different functions regarding grid visualization (this time i will consider buttons instead of menus, since the latter are seldom used for this purposes), namely

 *Zoom in*

 *Zoom out*

 *Zoom area*

 *Pan*

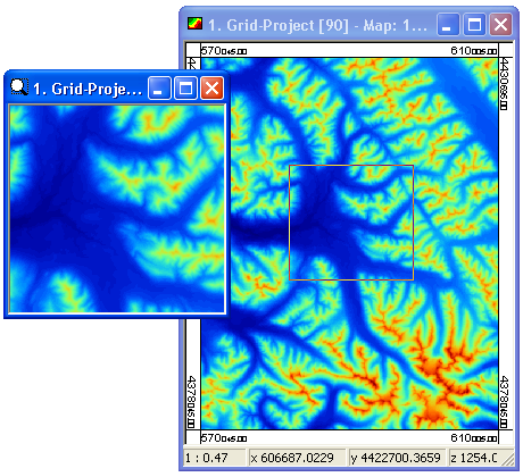
- The *Zoom in* tool will make the grid window bigger, thus "approaching" it to the user.
- The *Zoom out* tool will shrink the grid window, thus "taking away" the grid from the user.
- The *Zoom area* tool allows you to select the region that you want to view. Click and drag onto the image to define this region. This tool does not expand or shrink the grid window, but adapts the defined region to the actual size of it. The smaller the region you define, the closer you will see it, since it will have to be expanded to fit into the grid window.

  If you want to go back to a previous extent, right click on the image. If you select a small region and after that you use the *Zoom out* tool, the image will grow, but it will still be confined to the region defined by the *Zoom area* tool.

  You can return to the full extent by using the *Grid/Grid View/Full Extent* menu item.
- The *Pan* tool allows you to move the defined region along the grid. Simply click and drag to do it.

Another tool that can be used for this purposes is the magnifier. To activate it, select the *Magnifier* menu item under the *Grid* menu (or click on the *Magnifier* button)

You will see a little window next to the grid window. Do not maximize the grid window, since it will hide the magnifier. Place them one by the other. Now, if you click on the grid window, a bounding box will appear. The area enclosed by this box can be seen in the magnifier window at a greater scale. You can modify the size of this window and the bounding box will automatically adjust its size.



After using the magnifier window, simply close its window by clicking on its upper–right corner.

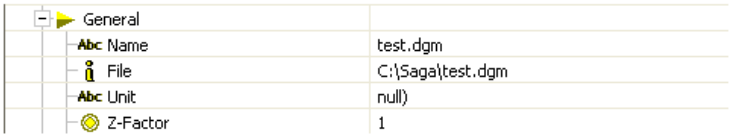
3.3 Adjusting grid rendering

Now that you know how to "move" through a grid layer, select an area and display it at your preferred resolution, we can see some further adjustments that will improve the way raster data is displayed. These, along with all the functions that we have previously seen, will allow you to fully interpretate grid data, and to get the most out of any grid layer that you use within SAGA.

To access all the aforementioned adjustments, select the *Grid/Settings...* menu item. This will cause a new parameters window to appear, containing many different parameters and also some additional non–editable information about the grid. The values in this window correspond to the grid that is selected (the one you worked with the last time). Since this values are not common to every grid, but can be adjusted specifically for each one, you can render each grid using different parameters depending on the kind of information they contain.

Another way to get to the grid parameters window is by right–clicking on a thumbnail in the project window and then selecting the *Settings* menu item.

The first parameters we find in the grid parameters window are some basic data about the grid itself. The only editable fields here are *Name* and *Units*.

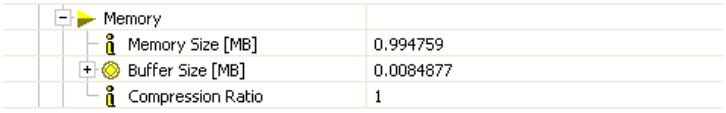


Modify the *Name* field to change the name used to identify the grid. Notice that, although the name of the grid and the name of the file where it is stored are usually the same, changing the former will not affect the latter.

Modify the *Units* field to specify the units in which the Z values are expressed. This unit will appear next to the Z values in the status bar of the Grid window.

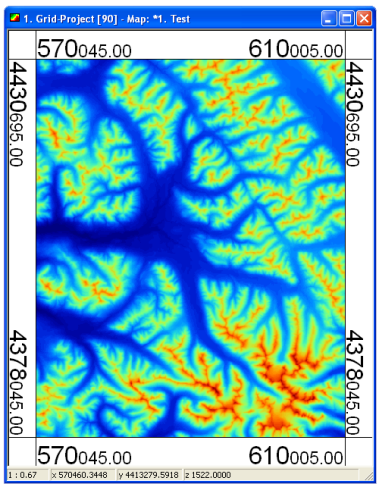
The cell value that appears on the grid view is the value contained in the cell multiplied by the value in the *Z factor* field. Change this if you want these values to be expressed in other scale or other units different to the ones really stored in the grid. We will see a usage of this in the chapter about terrain analysis.

After these parameters we find a small set of them related with memory handling.

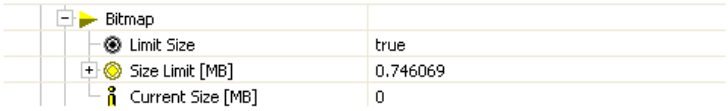


Again, they will not be explained here, but a little bit later in this same chapter. Anyway, safe for some rare cases, it's usually better to leave them unchanged and accept their actual values.

The *Frame Size* parameter controls the size of the frame area around the image, where the corner coordinates are displayed. In the following image you can see how it affects the way grids are displayed (notice the bigger frame size). This will also have effect when creating layouts.

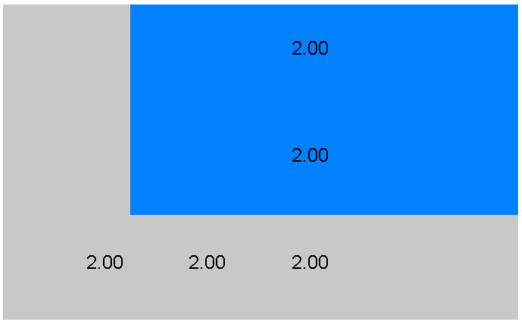


Unless you work with very large grids or you are running short of memory, you are not likely to use the next parameter, *Limit Size*.



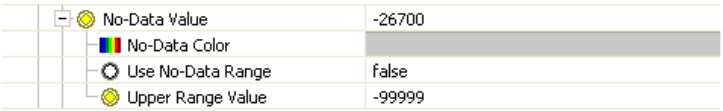
If you set it to true and enter a bitmap size in the *Size Limit* field, SAGA will not use more than this amount of memory to display a grid. That means that, if more memory is needed when you enlarge an image, it will not be shown in full detail event if the grid itself has enough resolution. Try setting a very low value and then use the zoom tools in the grid window to enlarge it. You will probably get a heavily pixelated image.

Again, this is not a feature you are going to adjust very often, but you must take care when opening large grids, specially if they contain feature information. For example, have a look at the following screenshot, which represents a close up of a grid containing road information. Cells with a value of 2 are roads, while cells without value are no–data cells.



Numerical information is not coherent with cell coloring!! (we will see in a couple of lines how to make those numerical values appear on screen). This is because SAGA has not enough memory to create a detailed bitmap and, if you zoom too close, you can notice this lack of detail. This same low detail is used when saving a grid as a bitmap, so you should not limit bitmap size if you plan to create an image from a grid.

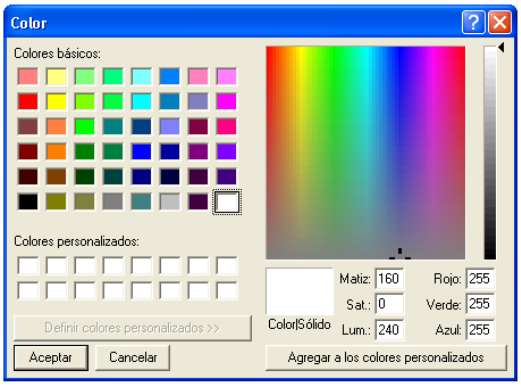
Going back to the parameters window, a more useful set of parameters is the one related with no–data values.



Sometimes, you do not have information in all the cells of a grid layer. SAGA provides a very easy way to tell the program how this cells can be recognized. When SAGA encounters one of this cells, it just ignores it and does not use it for any of the calculations at all. No data cells are usually identified by the value they have, which is usually outside the range in which "normal" values for the variable being represented can be found. For example, if you are working with a DEM (that is, you are working with heights) and using meters as units, a value of -99999 is not likely to appear, so can be used as a no data value. In fact, -9999 and -99999 are probably the most usual no data values.

To set this value, just enter it in the *No Data Value* field. In case you want to use a range of no data values instead of just a single one, set the *Use No–data Range* field to true and then introduce the upper range value in the *Upper Range Value* field. The value previously set in the *No Data Value* field will be used as lower range value.

No data cells (whether they have been identified using a single no data value or a full range of them) are represented with an special color. You can select it by clicking on the *No Data Color* field to see the typical Windows color picker dialog (mine is in spanish, as you can see...).



You can find some fixed fields (you cannot edit them) below the last ones, which provide some basic statistical information.

Value Range	
Minimum	937.15
Maximum	2442
Arithmetic Mean	1499.65
Standard Deviation	275.021

Since they cannot be modified, no more has to be said about them.

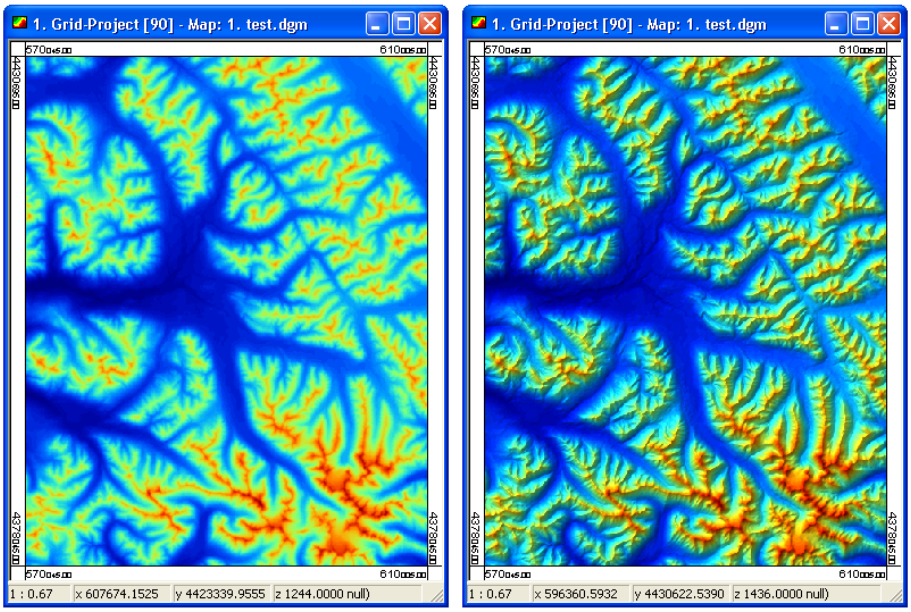
After the previous general parameters, we get into the *Colors* section, probably one of the most important ones regarding grids, and the one that we will use to modify the way a grid layer is displayed on screen.

Colors	
General	
Shade Grid with...	--- NOT SET ---
Mode	High Values - Dark Shade
Range (%)	
Minimum	-40
Maximum	60
Classification Type	Metric
Scheme	
Table	--- UNNAMED ---
Metric	
Color Classes	
Display Range	
Minimum	937.15
Maximum	2442
Scaling Mode	Linear
Logarithmic Scale Factor	1

The fields under *Shade Grid With...* can be used to add a relief appearance to any grid. The most important parameter here is the name of the grid that contains the shading information. These grids can be generated from a simple DEM by using the *Terrain Analysis/Lighting/Analytical Hillshade* module, which will be described in a later chapter. However, the demo.zip file you downloaded also includes a hillshade grid. Of course, to use a hillshade grid, it must have the same extension and cell size as the grid you want to shade. Therefore, if both of them are already loaded, they should be in the same project, not in separate ones.

Only those grids that are in the same project as the grid whose shading is being adjusted will appear on the list in the *Shade Grid With...* field.

The following picture shows the difference between the shaded and the unmodified image.

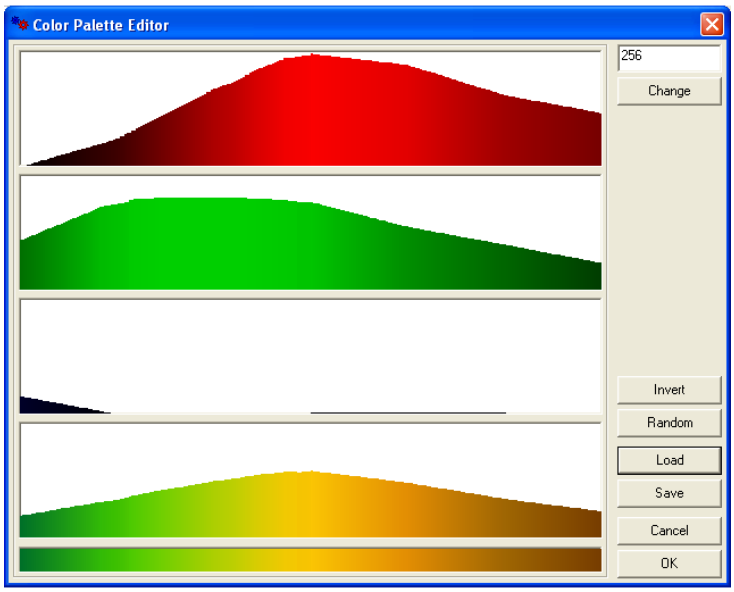


Adjust the intensity of the shading by modifying the *Mode* and *Range* parameters. I guess that the effect of this parameters can be better understood by "toying" a bit with them instead of trying to explain it, so just try yourself. Anyway, default values are usually better than any other combination, so you will probably go back to them after a few tries.

The *Classification Type* item can be used to set how SAGA displays cells with different values. If you set it to *Metric* (its default value), SAGA will define a color ramp from the value of the minimum value cell to the maximum one, and assign colors to the remaining cells (whose values will always be between the two previous ones) by simple interpolation. This method is recommended better when working with grids that represent continuous variables such as height, temperature or slope.

Instead of using the maximum and minimum values as boundaries for the color ramp, the *Display Range* field can be used to set user–defined values. If you want to use the relative maximum and minimum values of the displayed area (not of the whole grid), set to true the *Automatic Stretch* parameter, located near the bottom of the parameters list.

You can define the color ramp used, by clicking on the *Color Classes* field. A window similar to the next one will appear.



Four graphic boxes are shown in this window. The three first one correspond to the color components red, green and blue, while the last one represents the colors obtained by combining them. You can change this colors by clicking and dragging on each box. Again, just mess with it until you understand how it works, it is quite easy.

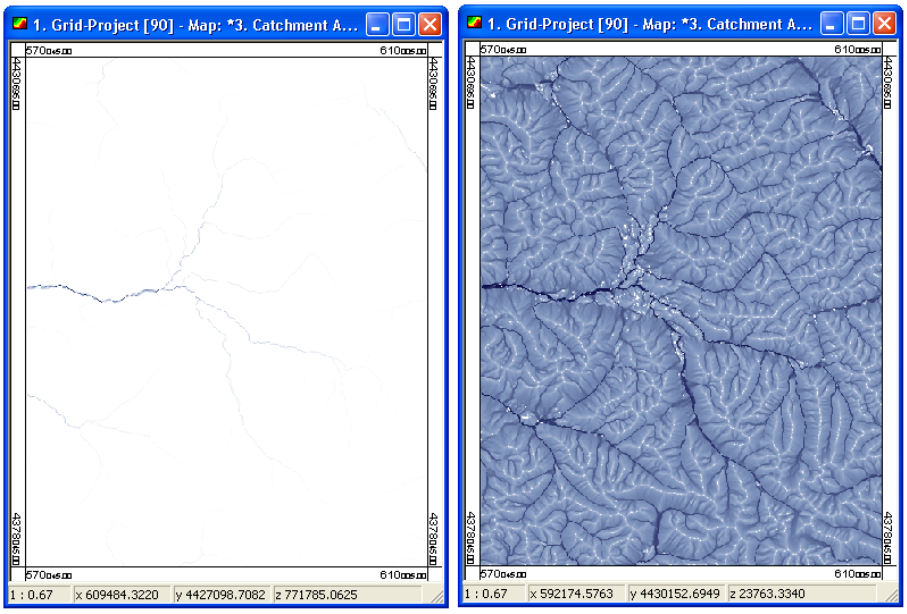
You can load and save color ramps using the *Load* and *Save* buttons. SAGA comes with a large set of predefined color ramps that can be found in the *pallets* folder. Have a look at them, you will surely find one that fits your needs. (I love the *eire* one . . .)

You can generate random palettes by using the *random* button

The level of detail of the palette can be modified by changing the number of classes it is divided into. Just enter this number in the textbox labeled *Classes* and then press the *Change* button.

The way SAGA performs the interpolation between the max and min values can also be modified. Linear interpolation is the default, but you can select logarithmic or inverse logarithmic interpolation using the *Scaling Mode* parameter. If using one of the two logarithmic modes, you can also set an scale factor using the corresponding field.

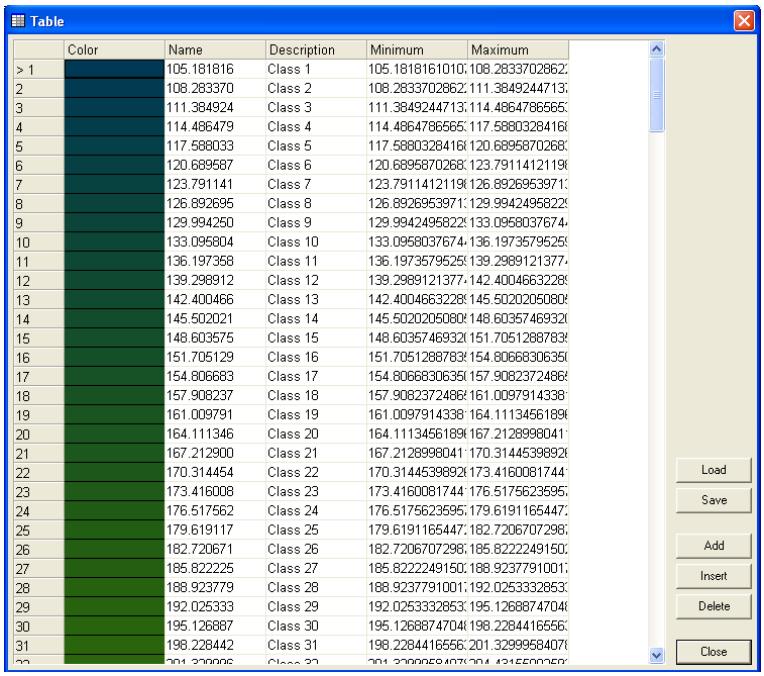
Logarithmic scaling mode is useful when displaying grids with a high standard deviation. For example, you will need it when dealing with flow accumulation grids, which will be a key tool in the chapters dedicated to terrain analysis and hydrology. By now, just have a look at how different a flow accumulation grid looks when displayed using a linear scaling mode — on the left, not a lot of information is shown — or a logarithmic one — on the right, with much more information being shown —.



The logarithmic scaling mode also affects how histograms, which will be studied a few pages later in this same chapter, are displayed. When we reach that section, we will explain this in detail.

If instead of a continuous grid you have a grid with discrete values (i.e. a land cover grid), maybe you would like to change the *Classification Type* from *Linear* to *Scheme*. If you do this, SAGA will not use a color ramp, but a table, to assign a color to each grid cell. This table contains different groups, each one of them with a lower and an upper value, and the color that has to be assigned to the cells with values into this group.

To define this table, click on the *Table* field. You will get a window with a table that represent the current palette, with as many different rows as classes were in the palette.



There is a chapter in this book dedicated to tables, so here I will just show some fundamental ideas.

You can change the four text fields simply by typing in them the values you want. Don't forget that the *Minimum* and *Maximum* fields should contain only numerical values.

Use the *Add* and *Insert* buttons to add new rows at the end of the table or at the current place. To delete a row, click on it to select it and then use the *Delete* button.

To change the colors, you have to double click on the color cell to get the color picker window.

This classification type is also useful if you want to make different groups when displaying a continuous grid. For example, let's say that you have a grid representing slope and you consider that slope under 10% is OK for a particular activity, slope between 10% and 25% is not so good but still suitable, and, finally, that this activity cannot take place in those cells whose slope exceeds 25%.

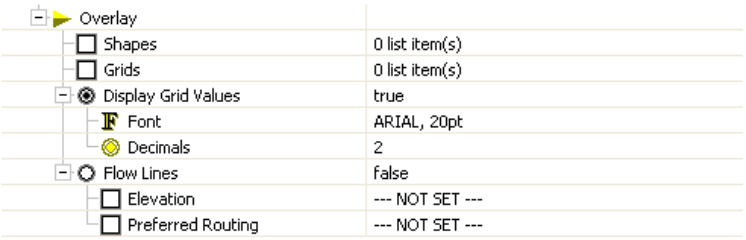
The table in this case will look like the one under this lines.

Try to create it by using the functions explained above.

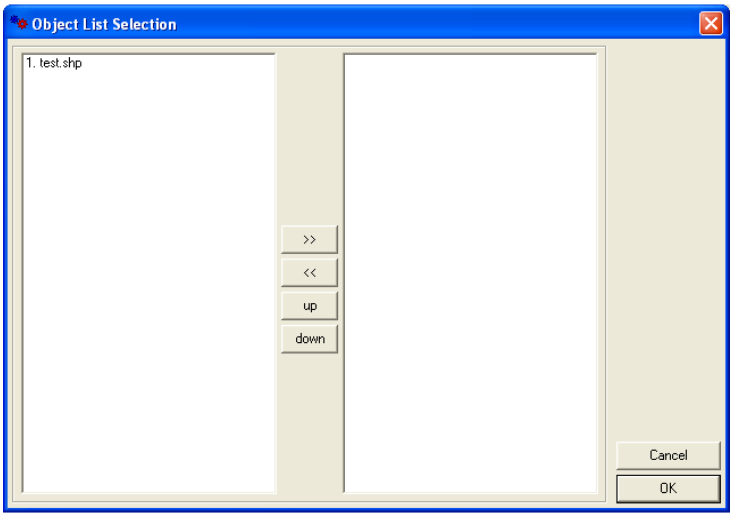
Notice that a similar effect can be achieved reclassifying the grid data into groups (this can be done using a module) which will turn the continuous information into a discrete one, but this will completely alter the data. If you simply change the table used to assign the colors, the grid will remain unchanged. We can say that reclassifying grid values is a "real" grid operation (and, as such, needs a module to be performed), while using a table (this kind of tables are usually called *look–up tables*) is just some sort of visual "trick".

After this brief explanation, let's continue analyzing other parameters found in the parameters window.

When representing a grid layer, it does not have to be represented apart from other grid or raster layers. We have already seen how to combine two grid layers to create a shaded relief image. Now we will see how to add vector elements and other grid layers and combine them into just one view.



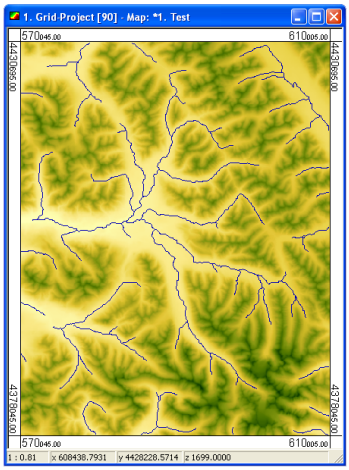
To add a vector layer, click on the *Overlay/Shapes* item and you will see the following window:



We haven't seen this window yet, but it will appear whenever SAGA permits a multiple selection of elements. On the left size of the window you have all the elements that can be selected. As you double click on them, they also appear on the right size, which contains the selected elements. Double click on the right side to remove elements. They can also be selected (or unselected) by single–clicking on any one of them and then using the >> and << buttons.

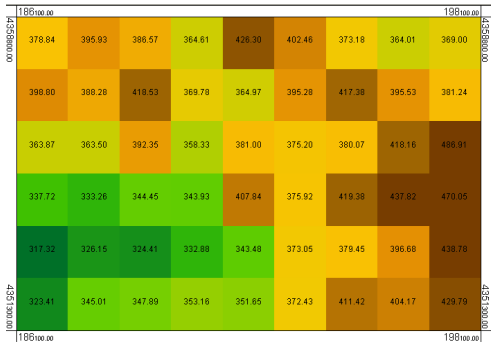
Using the *Up* and *Down* buttons, the selected elements can be ordered according to your preferences. This order will affect the way layers are displayed, in case they overlap. You will understand this when you get to the *Working with shapes* chapter.

The test.dgm grid with the test.shp file on it looks like this:

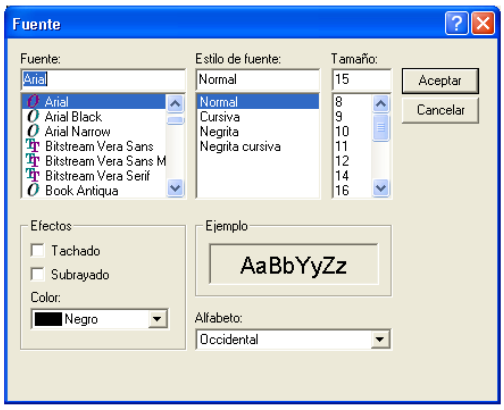


To add grids, the procedure is almost identical. Click on the *Overlay/Grids* button and discover it by yourself.

Setting the *Display Values* parameter to true will cause grid values to be displayed when you get close enough to a grid using the zoom tools, as shown in the following image.



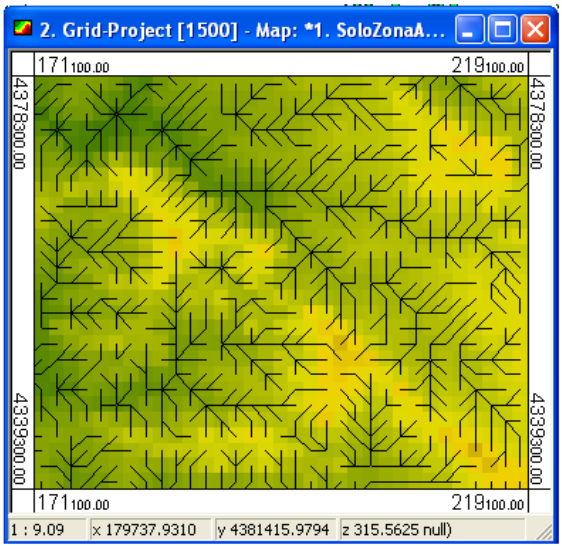
The number of decimal places and the font used to display the values, both can be adjusted using the *Decimal Places* and *Font* parameters. Clicking on the *Font* field you will cause the following dialog to appear (again, mine is not in English but in Spanish...).



Probably you will already know how to use it, since it can be found in almost every Windows application...

The last additional element we can incorporate into a view are flow lines. Select a DEM in the *Elevation* field and (in case you have one) a preferred routing grid in the field with the same name, and you will see how cells are connected using their flow directions. This will be all explained in detail once we reach the hydrology chapter, don't worry if you do not understand what it means.

When you get close to a grid, flow lines look like the ones in the image below.



However, if you zoom out of the grid, you will reach a point (unless your grid is really small) where flow lines cannot be distinguished. Depending on the scale you are using, you will like to activate this feature or remove it, since it is rather time consuming and flow lines can even hide the grid completely.

And finally, the last parameter in the Grid Parameters window allows you to numerically set the zooming factor instead of using the zoom tools.

Changes will be made once you close the parameters window and the view can be updated.

3.4 Memory handling. Saving grids

Once you load a grid, it is stored in memory and SAGA doesn't need to use the file to retrieve the information contained in that grid. In a typical SAGA session, you are likely to use a few "source" grids and, using modules, create a a few more. If the area covered by those grids is large or if you use a small cell size, each grid will contain millions of cells, each one with its own information. Consider this, and you will see that it is not so difficult to create a SAGA project that contains more data that what can be held in memory.

For this reason, a good memory management is a key element of a GIS software, and SAGA is no exception. SAGA incorporates compression methods that "shrink" grids so they don't need so much memory to be stored. You should use them in case you are having problems with memory. Otherwise, its better not to use them, since they cause data access data to be slower.

Also, if even more memory is needed, data can be stored in the hard disk using virtual memory resources.

To select the way a grid is stored , you can select the menu items *RTL compression* or *File Cache* in the *Grid/Memory* menu.

These concepts regarding memory handling are rather technical, and perhaps not suitable for the average SAGA user, so if you don't understand what those terms means, you probably don't need to use them.

Another much more useful set of features can be found in the *Grid/Save* menu. If you have modified a grid or created a new one, you might want to save it. As we saw when we loaded the test.dgm grid, some file formats are available without using any modules, while others are not. The same applies to saving a grid, since only the native formats (i.e, the DiGeM format) can be used to save the grids included in a grid project. Other formats are also supported, but we will not learn how to use them until we reach the chapter dedicated to Import/Export modules.

However, there are other interesting functions regarding grid saving that we will not found in any module, thus compelling us to employ the DiGeM format if we want to use them.

If you simply want to save a Grid, select that grid and then go to the *Save/Save...* menu item. Type in the filename you want to use, and your grid will be saved.

If you don't want to save the whole grid, but just a portion of it, you can use the zoom tools to display the desired portion and then use the *Save Zoom Area...* menu item. You will not find this function in any module, so if you want to export the zoomed area to any other format supported by SAGA Import/Export Modules, you will have to save it as a DiGeM file, then open it, and finally export that whole file in the desired format.

To create a new project restricted to that area, you can use the *Save Zoom Area as new Project* menu item.

3.5 3D views

A favorite of many users (though seldom used for "real" practical purposes), 3D–views offer a new way of displaying data, helping you to be aware of many details than can be overlooked when just using simple 2D representations. Also, they are fun (at least the first times you use them), and quite impressive, so it is worth writing a couple of lines about them.

To make a 3D view, you need two things: a grid to represent and another one where the height values are stored. Of course, both of them can be the same. Also, heights does not have to be "true" heights (that is, heights over sea level), but any other parameter than you might want to see in 3D. For example, a 3D view using a accumulated cost surface (we will see how to create them using the cost analysis modules) as height grid can be very useful to understand where least cost paths must be located.

To see a 3D view of a grid, right–click on its thumbnail and select the *3D–View* menu item.

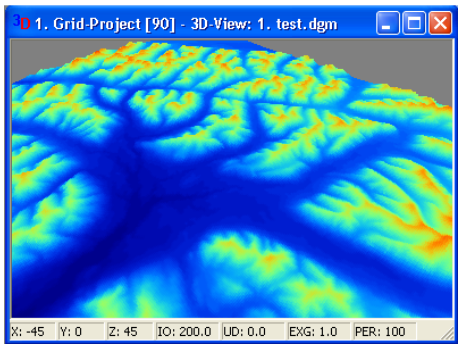
The parameters window you will see has many different parameters you can set. However, most of them can be changed more easily once you are in the 3D rendering screen, so we will ignore them now and just consider those that have to be set from this window.

The most important one is, of course, the elevation grid. As it happened when shading a grid, only those grids with similar dimension and cell size as the one been represented can be used as elevation grid. You will find them in the list in the *Elevation* field.

Skipping most of the remaining parameters we get to the two last ones. Using them you can change the appearance of the background of the window, setting it to a single color (you should already know how to select a single color as a parameter in SAGA) or a bitmap. When you select a bitmap, the background color is ignored.

Once this parameters have been configured, click on the *OK* button.

The 3D–view window should look like this.

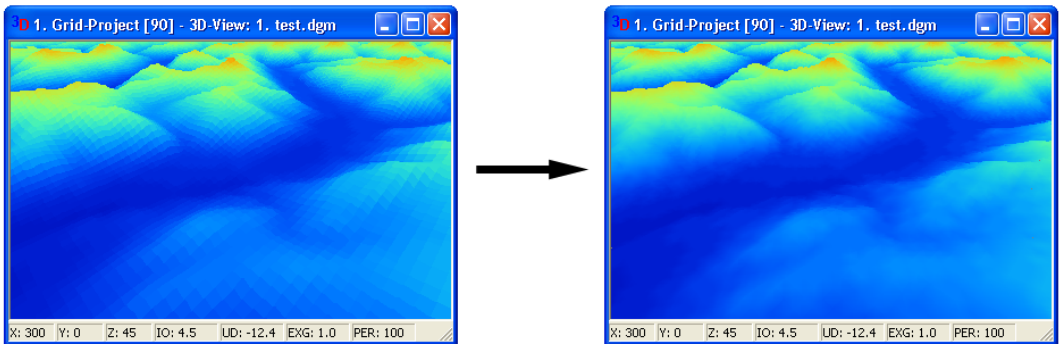


If you ever wondered what it was like to fly through the Yukon valleys like an eagle but never could do it, this time you will be able to make your dreams come true. Hold on... we are taking off! Once airborne, grab the mouse and drag over the 3D–view while right clicking. Then, left–click and drag it to alter the rendering in a different way. These are the two main controls you have to modify the view and adjust it to your needs. As you click and drag, the values in the status bar of the 3D–view window will change. Now, if you select the *Settings* menu item under the *3D–View* menu, you will see the parameters window again. Notice how the rotation and shift values have changed. Close the window and go back to the 3D–view window.

By using all the arrow buttons in the toolbar you can move an rotate the surface in many different ways. Again, the behavior of these functions is better understood simply by using them and experimenting, so I will leave them unexplained.

The buttons that control the vertical exaggeration (the two right–most ones) are specially interesting, specially when using an elevation grid with a smooth relief.

If you bring the surface too close to you, you might be able to distinguish the pixels, revealing the poor resolution of the image (at least, for that scale). To avoid this, SAGA can interpolate the color values between adjacent cells to smooth the resulting image, as it is shown under these lines.



This task is rather time consuming, so you might want to disable it with larger grids or if you don't have a computer with a large amount of memory.

Once you have obtained a good perspective of the surface, you can save the image so later you can include it in another document or do with it whatever you want. Select the *Save Bitmap...* menu item and type in the filename you want to use. Unfortunately, only bmp files are supported, but it is quite easy to convert them to JPEG format or any other popular format, using a more specific image software.

Apart from saving single static images, you can use the built–in sequencer to save sets of images that can be later used to create fly–through animations.

Before starting the sequencer, select the base name that will be used to compose the full file-name under which each frame of the sequence will be saved. Select the *3D–View/Sequencer/Save Frames...* menu item and use the dialog that will appear to get to the folder where you want to save the frames. Then, type the base name, for example sequence. SAGA will save each frame using a filename like sequence_001.bmp, sequence_002.bmp ... and so on.

Now, activate the recorder selecting the *3D–View/Sequencer/Record* menu item. Once it is activated, you can use the usual mouse control to move through the 3D–view and change its rendering parameters. As you change the view, SAGA will record the corresponding frames in the folder you specified.

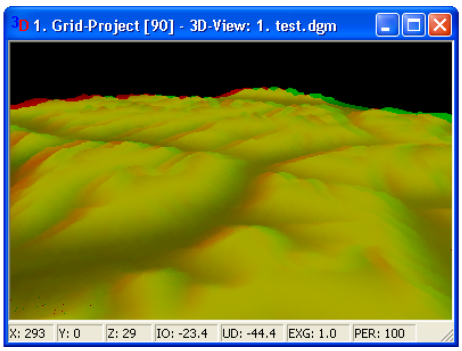
When finished, click again on the *Record* menu item to stop recording.

You can add a frame with the current view to the sequence by using the *Add Position* frame.

Frames are stored in bitmap format in your hard drive, but SAGA also keeps them in memory while you are in the 3D–view window. You can retrieve them and see a fly–through animation using the *Play* command in the *Sequencer* menu.

If you want the animation to repeat over and over again, select the *Play Loop* menu item.

The last (and, definitely, the most bizarre) function regarding 3D–views is the creation of anaglyphs. Yes, you are right, anaglyphs are those things that, when you look at them using glasses with each eye of a different color, you can see them in full 3D. I have to confess that I have not tried this completely (it is not very easy to get the glasses right now, and I am too lazy to build a pair of them myself), but here is a screenshot of an anaglyph created using SAGA so you can have an idea of how they look like.



Some parameters related with anaglyph creation can be adjusted using the *Settings...* menu item, but I am afraid I cannot write much about them.

To finish this section, here is a little tip: to get more realistic 3D views, shade the grid before using it to generate the view.

3.6 Basic analysis functions

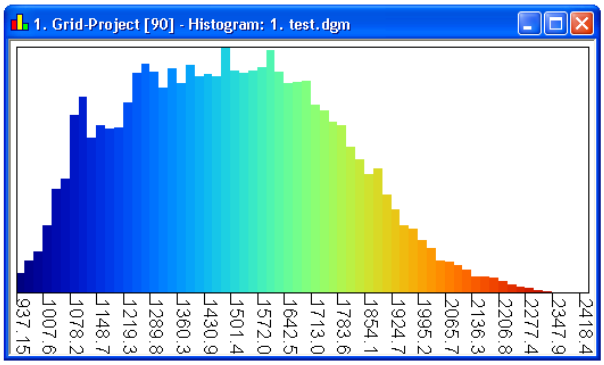
Time has come now to start obtaining our first results. In this section we will get some graphical results, but also some numerical ones. Although they are not very numerous (I repeat that most of the SAGA functionalities are kept in additional modules), they will help you to fully understand the SAGA GUI and the SAGA structure, and will prepare the path to the *Working with shapes* chapter (there are many things in common between vector functions and raster functions in SAGA) and, later on, to the detailed study of modules. The results we will obtain include:

- Frequency histograms
- Profiles
- Regression analysis

3.6.1 Frequency histograms

To know how values are distributed in a grid, you can generate a frequency histogram from them. To do it, select the *Histogram* menu item in the *Grid Menu*. As it happens with other grid commands, you can also find this menu item in the menu that pops up after right–clicking on the grid thumbnail.

The frequency histogram for the test.dgm grid looks like this:

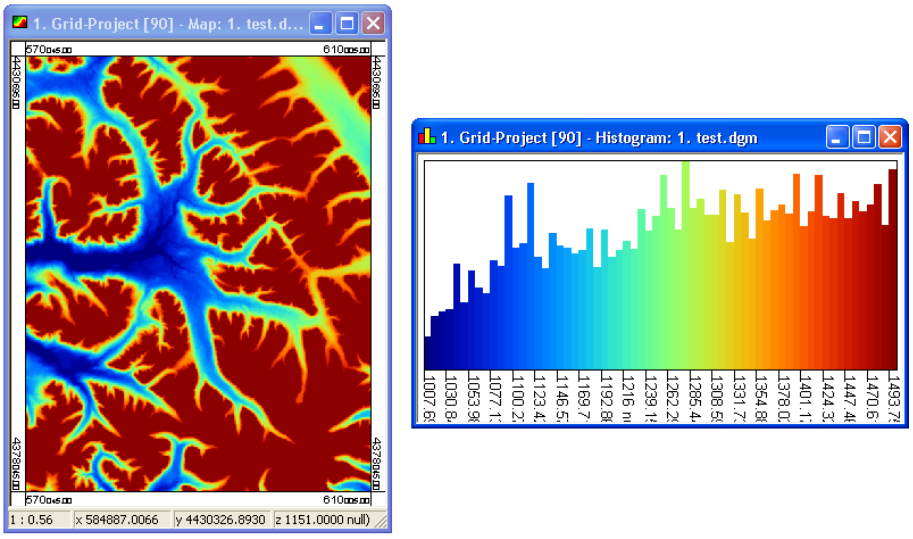


In it, you can see how height values are distributed in the grid, ranging from a lower value around 950 meters to a higher one around 2400 meters, with most values not being greater than 2000 meters. Also you should be able to recognize the typical bell–shaped curve, since those height values follow (more or less) a gaussian distribution.

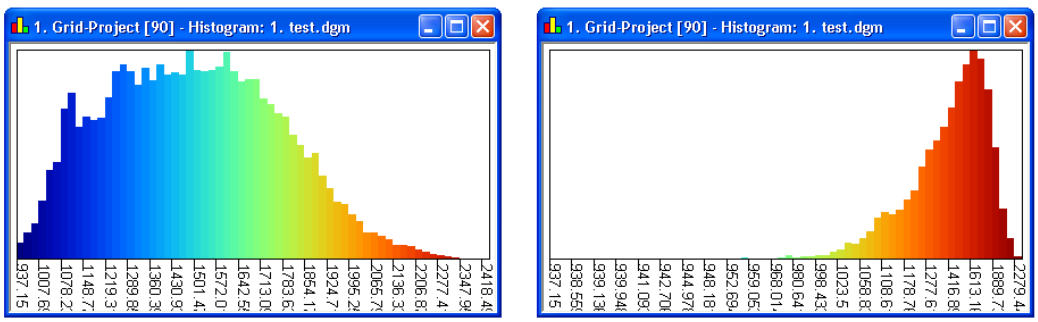
Depending on the kind of information that is represented in the grid, histograms might provide further information regarding other different variables. For example, the histogram derived from a time–out grid resembles the unit hydrograph of the basin considered for calculating those time values, thus providing some hydrological information about that basin.

What else can we do with a frequency histogram apart for just looking at it and interpreting its meaning? Well, first of all it can help us to modify the range of values used to display the grid and assign a color to each grid cell. Imagine that, for some reason, you want to graduate the values from 1000 to 1500 meters, and neglect those ones that fall outside those limits. Of course, you could use a table instead of a color palette, or even reclassify the grid data, but it is quite easier to change the rendering range. You can set the values numerically in the grid settings window, or directly on the histogram window just using your mouse.

To define a new range, just click and drag to select the values contained in it (in this case, those between 100o and 1500 meters. When you release the mouse, the grid view and the thumbnail will be updated. Every cell with a value lower than the lower value of the range (1000) will be displayed using the lower color, and those cells with higher values that the higher range value(1500) will get the higher color.



If you change the classification type, the shape of the histogram and its coloring will also change, as can be seen in the following two histograms (linear classification type on the left, logarithmic on the right) obtained from the test.dgm grid.



Histograms are not just a visual element. Their source information (the one used to draw the histogram) can be shown as tabular data. To do that, select the *As Table* menu item, (the only one) under the *Histogram* menu.

We will work more intensely with tables in the next chapter and you will learn how to handle them within SAGA. By now, just remember how to get a table from a frequency histogram, because we will use these tables later.

3.6.2 Profiles

Imagine that you want to go from point A to point B, and you have the DEM of the area in which those points are located. You might like to know if the line that connects both points (or even a path made up of several linear sections) hopefully goes through flat terrain or you will have to climb steep slopes in your way. A profile of the path would be a very interesting tool for that purpose.

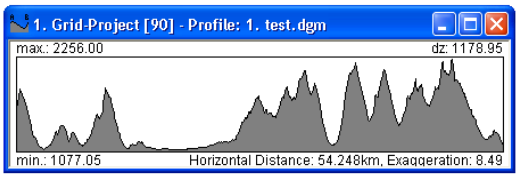
Now imagine that you have created an accumulated cost surface (you will learn how to do that in a few chapters) and you want to a least cost path using it. Also, you are interested in having the longitudinal profile of that path.

For the two preceding cases, SAGA features some simple (yet powerful) capabilities for dealing with profiles, which will be explained in this section.

There are two main ways to define a path: drawing it as a set of connected linear segments, or defining a single point and letting SAGA calculate the rest of points according to some conditions. The former is the one selected by default, so I will start with it.

To define a profile, select the *Profile* menu item under the *Grid* menu. It will only be available if the view window of the current grid is visible (you have to use it, you cannot define the path on the thumbnail!).

A blank window will appear. As you start introducing points, the profile along the lines defined by those point will be drawn in this window.



To introduce a point, simply click on the grid view window in the location where you want to place that point. After you introduce the first one, when you move the mouse over the grid, you will see a line joining the current mouse position and the last point you introduced.

Along with the profile itself, additional information can be found in the profile window, namely:

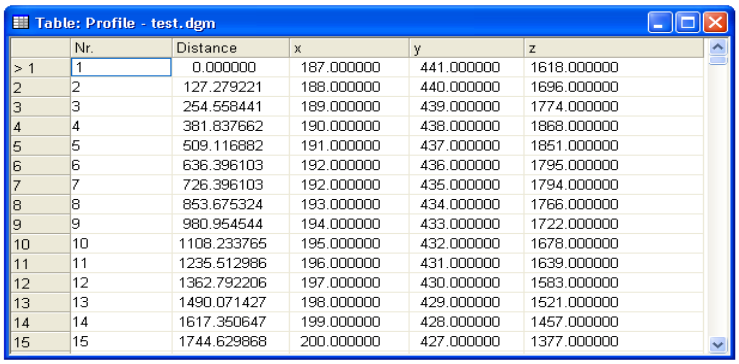
- Vertical exaggeration
- Horizontal length of profile
- Maximum Z value
- Minimum Z value
- Difference between the two above factors.

Select as many points as you want and then right–click to finish the path.

Now you have the profile you wanted (in case you want a different one, just start creating a new one an it will be replaced). To suit the information in this profile to your needs, you can adjust how it is displayed or you can convert it to a different data format (right now you just have an image of the profile). You can convert it to a table, a point vector layer (one point for each cell the path runs across) or a line vector layer.

To do this conversion, select one of the menu item under *Profile/Convert to ...*. Don't forget that you have to "activate" the profile window (just clicking on it) to see the *Profile* menu.

If you select *Points* or *Lines* a new window will appear containing vector data. If you select *Table*, you will see a table like the following one.



	Nr.	Distance	x	y	z
> 1	1	0.000000	187.000000	441.000000	1618.000000
2	2	127.279221	188.000000	440.000000	1696.000000
3	3	254.558441	189.000000	439.000000	1774.000000
4	4	381.837662	190.000000	438.000000	1868.000000
5	5	509.116882	191.000000	437.000000	1851.000000
6	6	636.396103	192.000000	436.000000	1795.000000
7	7	726.396103	192.000000	435.000000	1794.000000
8	8	853.675324	193.000000	434.000000	1766.000000
9	9	980.954544	194.000000	433.000000	1722.000000
10	10	1108.233765	195.000000	432.000000	1678.000000
11	11	1235.512986	196.000000	431.000000	1639.000000
12	12	1362.792206	197.000000	430.000000	1583.000000
13	13	1490.071427	198.000000	429.000000	1521.000000
14	14	1617.350647	199.000000	428.000000	1457.000000
15	15	1744.629868	200.000000	427.000000	1377.000000

We will see how to work with vector and tabular data in the next two chapters.

Although this kind of profiles are useful, it is usually more interesting to let SAGA define the path from a single point. Also, this profiles, since it's the program who traces the path to follow, are far more precise (you can notice this difference when converting them to lines)

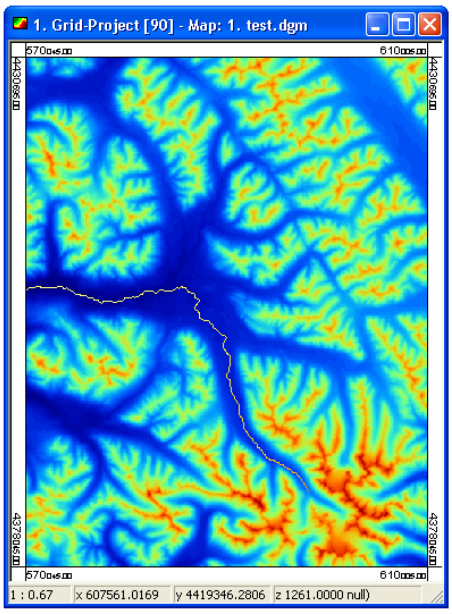
To create this profiles, select the *Trace flow* menu item. Now, when you select a point in the view window, SAGA will calculate the route that flow would follow downward from this point. This can be used to draw a profile of a river, going down from its header to its lower part.

SAGA traces the flow until it reaches the boundaries of the grid or a sink cell is found.

Now select the *Profile/Settings* menu item to see the profile parameters window. If you have a preferred routing grid, you can select it in the field *Sink Routes* an SAGA will be able to trace flow over sink cells.

You will probably remember that we mentioned preferred routing when we saw the *Flow Lines* parameter. As then, don't worry if you don't understand all this, because it will be explained later with much more detail.

The flow path will be drawn onto the grid view window.



When you want to define the route that flow will follow, you have to take into account the distances between each cell and its surrounding ones, so as to correctly calculate the slope that is found from the center cell to each one of its neighbors. However, if you are working with an accumulated cost surface, distances were already used to crate that surface, so you must ignore them. You will find a field named *Consider Distance*, that you should use to adjust the behavior of the routing algorithm in each case.

You can modify the way SAGA renders the profile, adjusting its exaggeration. It's usually a good idea to exaggerate the representations, specially when the relief is smooth.

There are two ways of selecting an exaggeration factor: using a fixed one or letting SAGA adjust it automatically to the size of the profile window. If you chose the latter (the default one), modifying the size of the profile window will cause the profile to be updated and drawn again using a new exaggeration factor (try it yourself...)

3.6.3 Regression analysis

Given two grids, you can use SAGA to check if there is any kind of correlation between them. Using the *Grid/Regression Analysis/New...* command, SAGA performs a cell–by–cell comparison between both grids and tells you how the variations of the values in each one are mutually dependent.

Let's see what that means. Open the testslope.dgm file included in the demo.zip file. You should have also the test.dgm file already loaded. While our well–know test.dgm file contains elevations, testslope.dgm contains information about the slope in each cell. This grid has been calculated using the terrain analysis modules explained later in this book.

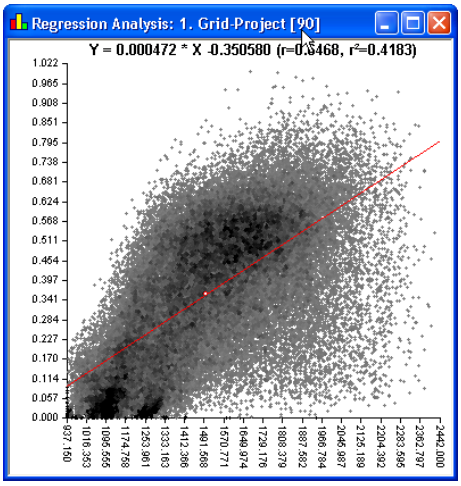
We are going to see if there is any relation between elevation and slope (that is, if steep slopes are usually found in higher areas and flat surfaces on low ones, or viceversa). To do that, select the *Grid/Regression Analysis/New...* menu item.

In the parameters window, set the test.dgm grid as grid A and the testslope.dgm grid as grid B (reversing this will just cause the X and Y axis to be changed). Leave the *Method* field set to *Grid A with Grid B*.

If the grid is too big, drawing all the value pairs might cause SAGA to get slow and take too much time in displaying the regression analysis resulting graph. You can modify the *Draw each x.tuple* to avoid this. The bigger the number you type in, the less points SAGA represents

and, consequently, the less time it takes to draw to display the graph. The default value (4) is a good choice for the grids we are using (unless your computer is too slow...)

Press the *OK* button and you will see something like this:



The classical r and r^2 values are displayed along with the scattered points, and also the equation of the best fit line (shown in red).

Chapter 4

Working with tables

4.1 Introduction

Tables are a very powerful tool for displaying data. Tables are important because they constitute a linking element between SAGA and other application such as spreadsheets.

Not all SAGA modules or functions generate new grids as result. Some of them return tabular data instead of new grids, and you should know how to handle this kind of result. Also, tables are required sometimes as input, and appear quite often in a regular SAGA session.

Most of all, tables are used to show data from relational databases, and those are tightly linked with vector data. That's why I have chosen to include this chapter before the *Working with shapes* one, since I consider them a very important part for correctly working with many elements implemented in SAGA.

4.2 Opening and editing tables

To start working with tables, let's open one of them. SAGA supports DBase files and delimited text files without needing any additional module. Thus, to open a table, just select the *File/Open...* menu. There is a text file included in the demo.zip file (named table.txt). Select it and press *OK*.

A new window like the one below should appear.

	City	Code	Population
> 1	Madrid	MAD	5086635
2	Barcelona	CAT	3765994
3	València	VAL	1397809
4	Sevilla	AND	1180197
5	Bilbao	PAI	947334
6	Málaga	AND	775458
7	Zaragoza	ARA	638535
8	Murcia	MUR	503568
9	Granada	AND	440302
10	Oviedo	AST	425829
11	Vigo	GAL	412939
12	Cádiz	AND	400157
13	Alicante	VAL	380357
14	Valladolid	CLE	368890
15	Tarragona	CAT	325333
16	Córdoba	AND	314034
17	Pamplona	NAV	285671
18	Gijón	AST	279837

You can change the information contained in a cell simply clicking on it and then typing the new value for that cell.

Clinking on the name of each column will cause the table to get ordered by the values on that column.

If you want to add a new row (in the case of the above table, adding a new city), you can use the *Add Row* menu item from the *Table* menu. if you want the new row to be placed not at the end of the table, but at any other point, just select the row above which you want to place it and then use the *Insert Row* menu item instead. (remember that we have already seen this commands before, when defining look–up tables for grid coloring)

You can delete rows as well, by using the *Delete Row* menu item.

Once the table has been modified, you can save it using the *Tables/Save* menu item.

4.3 Creating a table

Sometimes you might need to create a table from scratch. That is useful, for example, if you want to create a point layer from a coordinates file. You can create that file using a spreadsheet (and you should if the table is too complex or feature any kind of relation between cells, for SAGA has limited capabilities), but if the table is simple, you can use SAGA built–in functions.

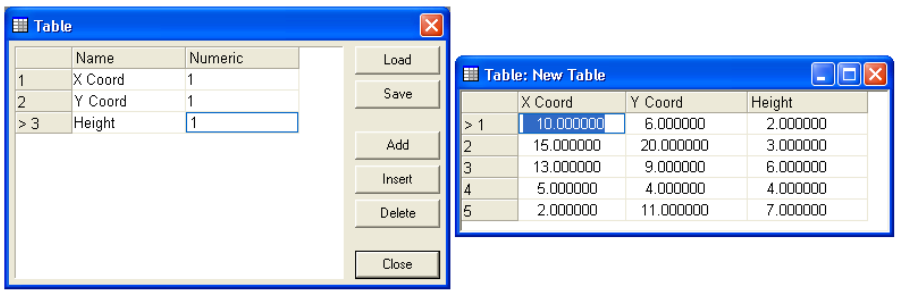
To add a new table to the Tables project window, right–click on it and select the *Project/Add New Table* menu item.

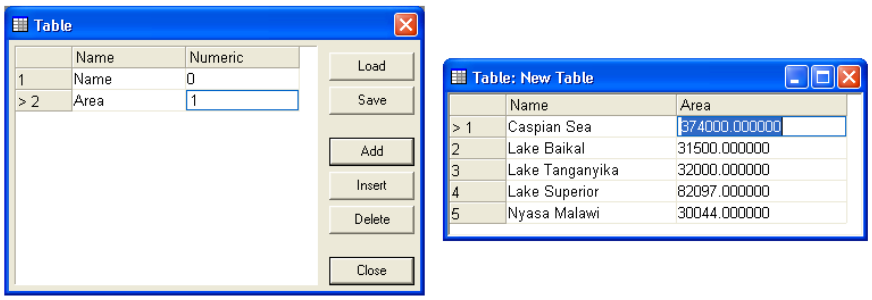
You will see a new table (although it has no rows at this moment), that you have to use to define the characteristics of the table you are about to create.

Use the *Add* and *Insert* buttons to add new rows. Each one of them defines a field in the table you want to create. For each row, you have two columns, one named *Name* and the other one named *Numeric*. Type in the name of each field in the *Name* column. Introduce a zero value in the *numeric* column if the field is to contain text information, or a non–zero value if that field will contain numerical information.

Add as many rows as fields you need in your table and then press the *Close* button. Your new table has been created and now you can start editing it.

Easy as it is, many people find this procedure a bit unclear. To help you better understand how it works, I have included a couple of examples in the following pictures. The images on the left show how data should be defined in the first table window, while the ones on the right show those tables once created and populated with data. That should give an idea about how these two tables (the "real" one and the one used to define it) relate to each other.





Try to create yourself the table in the first example, since we will later use it for some further examples.

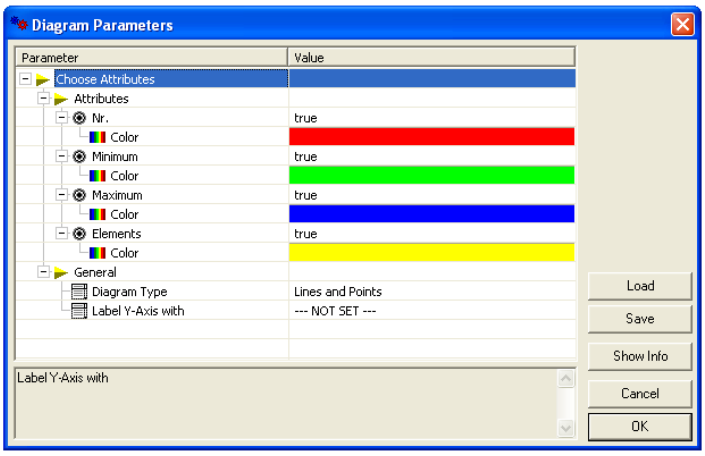
4.4 Creating diagrams

You can grasp a better understanding of the meaning of tabular data by graphically representing it. Although this can be done using a spreadsheet (surely, you have done that before), SAGA features some neat graphical functions that will suffice in most cases.

To see how his works, we will need some data. Instead of using data from the table we loaded at the beginning of this chapter, we will create a new one. If you remember, frequency histograms could be converted into tables. Create an histogram of the test.dgm grid and then convert it to a table (if you can't remember how to do that, have a look at the *Working with grids* chapter)

Once you have the table, select the *Show* menu item under the *Table/Diagram*

The following parameters window will appear.

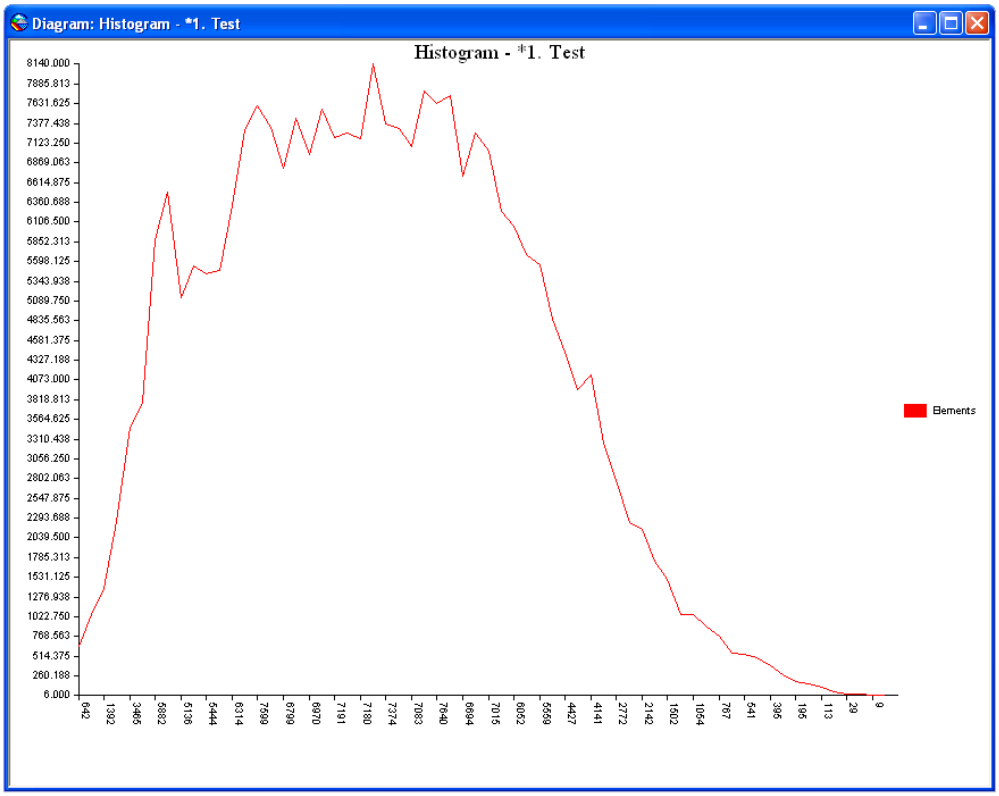


The *Attributes* node contains as many elements as fields (i.e columns) are in the selected table. Thus, it will vary depending on the table, so maybe the next time you try to create a diagram you will see a different number of parameters.

For each field, you can chose to represent it (set to true the parameter with the name of the field) or not (set it to false). If you choose to represent it, you can pick up a color for it using the *Color* parameter. The only field that is worth representing in the histogram table is the one named *Elements*, so you should set the remaining ones to false.

Use the *Diagram type* field to choose the type of diagram to create (select *lines* this time). In the *Label Y-axis with*, select the field to use for the scale that will appear along the Y-axis.

After all that, click on the *OK* button and you will get a diagram like the following one (I have changed the color from yellow to red. I found yellow too light for including the image in the text).



If you want to practice a bit more, do the following: Create a profile and then convert it to a table. Make a diagram with that table and you should get again the same profile.

Have fun!

Chapter 5

Working with shapes

5.1 Introduction

Once we know how to work with raster layers, it is time to start working with the other main type of spatial data: vector layers. As you have already seen, SAGA features all the functionalities that one needs to work with grid layers, and this functionalities are very easy to learn and to use. However, the most important grid functions will be shown when we study the grid analysis modules but, nevertheless, those are functions that are not really related to the handling of raster layers, but merely to the processing of them.

Handling vector layers is a bit different. Even if you don´t need to perform complex analysis using you vector layers, you will need a large set of functions to organize your layers, edit them, or even create them from scratch. Why is that? Mainly because of the different nature of vector layers and because this nature is a dual one. Let me explain you this duality. While grid data is contained in the grid itself, vector data needs to be stored in a database. This causes a vector layer to be separated in two different entities: the "shapes" (points, lines, polygons...) where the spatial information is kept, and the database where the information about those shapes is stored.

Large databases require large (and complex) Databases Management Systems (DBMS) and, unfortunately, SAGA lacks of them. You can use vector layers and open their associated databases, but many of the most important operations that can be performed with them are not available in SAGA. That is not so bad when dealing with small vector layers, but it is quite common to find vector layers that contain thousands of polygons and cannot be properly handled without the support of a strong and reliable DBMS.

Consequently, if you plan to develop your work mostly with large vector layers, SAGA might not be a good choice for you. However, the combination of vector data and raster data has many advantages, and SAGA is perfectly capable of combining them and making it easy for the user to handle both data types without effort. For example, if you want to create a DEM from point data by interpolating the height information associated with those points, SAGA will have no problem at all and you will not feel "strange" when using vector data (although you will have a somehow restricted freedom to directly work with it). In contrast with other "raster oriented" GISs such as IDRISI, where vector data support is rather awkward, SAGA raster–vector integration is seamless, such as the one found in, for example, ArcGIS (In fact, I would say that SAGA is just the opposite case to ArcGIS, because ArcGIS is mainly targeted at vector data and has some raster capabilities — though not as many as SAGA — implemented in the Spatial Analyst extension)

Enough or not for your work, SAGA vector functions are worth a chapter in this book. I guess that the main role of vector data within SAGA is to "serve" the raster layers and add some extra capabilities to the ones that could be implemented if only raster data were

supported. With that in mind, it the following pages I will try to cover all the tasks that can be accomplished using SAGA and regarding vector layers, including their creation and edition.

5.2 Displaying vector data

You have already seen how SAGA displays vector data. When you loaded the test.shp file containing the drainage network extracted from the test.dgm grid, it automatically opened a window where that drainage network was displayed. To follow the explanation, you should have this file loaded, so in case it is not, go and open it.

To learn a little bit more about how vector data is rendered on screen, let's open a new file. Go and open the test2.shp file. Instead of lines, this one contains polygons which represent the basins associated which the previously loaded drainage network.

You can notice how SAGA puts both vector layers in the same window, as opposed to raster layers which are kept in isolated windows. Due to this, some new buttons are found in the toolbar that allow you to change the view according to which layer (or which shapes within a layer) you want to consider each time.

You will see the *Zoom Area* and *Pan* buttons (I suppose you already know how to use them) and, on their right, the following new ones:

Zoom to full extent

Zoom to selected layer

Zoom to selected shapes

Zoom to previous extent

They are pretty self–explanatory but, anyway, i will explain them a bit.

- If you use the *Zoom to full extent* button, the zoom will be adjusted so all shape layers fit into the view window.

- The *Zoom to selected layer* will adjust the zoom so the selected layer fits to full extent into the view window. To select a layer just click on its name in the Grid Project window. The test and test2 layers they both represent the same geographical area, so the effect will be the same whichever one you chose. Also, since no other vector layer is loaded, there is no difference between using the *Zoom to full extent* and the *Zoom to selected layer* button.

  To depict a more realistic situation, imagine that you have a layer representing rivers of Germany and another one representing cities of Spain (no, I will not use those dreadful USA examples. I leave them for the ESRI guys...). If you just want to see the spanish cities, select the corresponding layer and then click on the *Zoom to selected layer* button.

- The *Zoom to selected Shapes* will let you adjust the view not to a whole layer, but to just some of his elements. (we will shortly see how to select elements). This is a very useful tool, but since SAGA lacks some of the most important methods for selecting shapes, you are not likely to use it as often as you would like to.

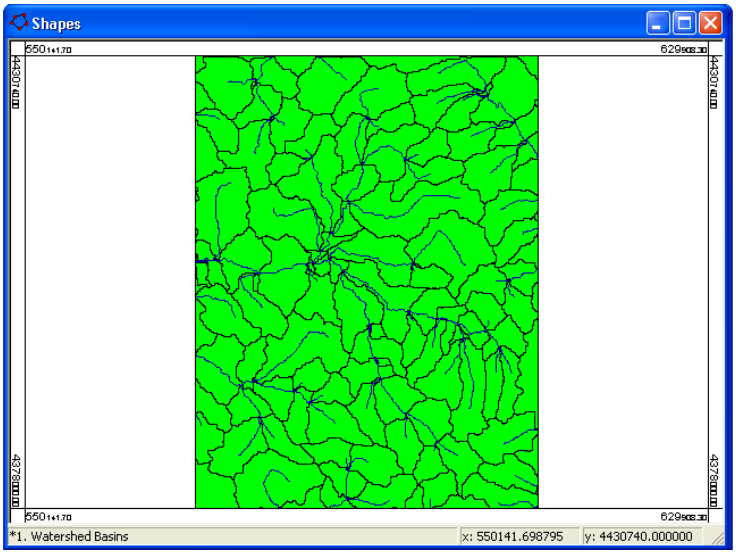
- I guess to *Zoom to previous extent* button does not require further explanation.

While you experimented with the above tools, you will have probably wondered where the drainage network was, since only the basins seem to appear in the view window. As both layers are displayed in one window, the basins are hiding the rivers because they were loaded later and, therefore, are rendered once the rivers have already been drawn. How can you avoid this? Well, imagine that you have to draw those basins and rivers with pencil and paper (you might need some crayons as well). What would you draw first? Of course, the basins, and then you would lay the rivers on them. If you have a screen instead of a paper, you still have to do it that way.

To tell SAGA that the order in which it renders the different layers have to be changed, you should use the display order commands. You can reach them by right–clicking on the name of a vector layer in the project window, and then going to the *Display* menu. Here you find the following menu items

- Move up
- Move down
- Move to top
- Move to bottom

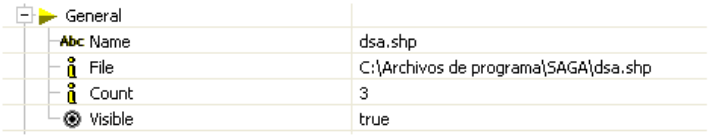
Use them to put the test2.dgm layer below the test.dgm one, and you should get a view like the one shown next.



Sometimes you might want to hide a layer without having to unload it. To do that, right–click on the layer name and select the *Visible* menu item. The tick (and also the layer) should disappear.

Like grids layers, vector layers have each one its corresponding parameters window. Again, you can reach it using the *Settings...* menu item under the *Shapes* menu. The fields in the parameters window control how each vector layer is displayed and, as we did with grids, we will explain them one by one.

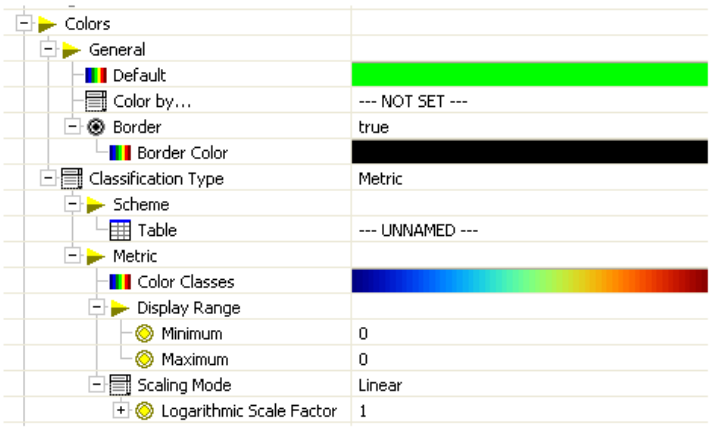
Among the ones under the *General* node, some are editable and some are fixed.



You can change the name of the layer using the *Name* field, and hide it setting the *Visible* field to *false* (though it is by far easier to do it without using the settings window.)

The *Count* field contains the number of different shapes that are included in the layer.

Again, the *Colors* node is the key one, containing the most important parameters.



Each element (whether point, line or polygon) has to be assigned a color. You have two options to assign this color: use a single color for all the elements in a layer or use a set of them.

If you choose to use a single color, select it in the *Default* field and then leave the *Color By...* field set to *— Not Set —*.

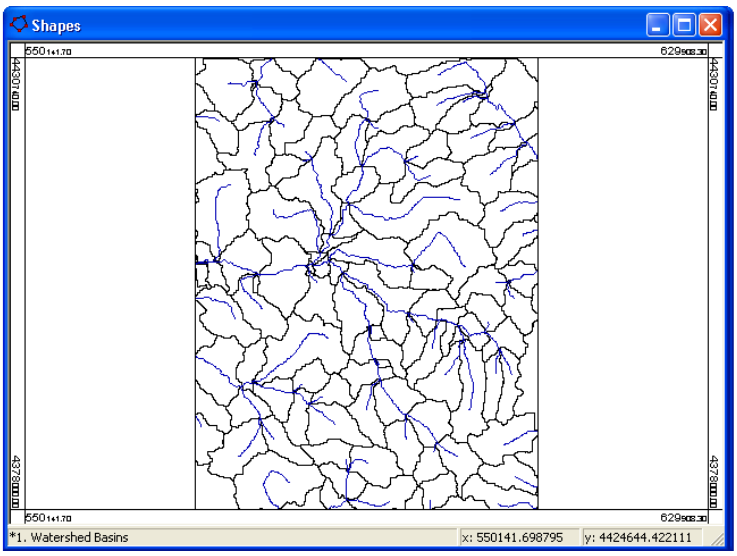
In case you want to use a set of colors, you can define it using a color palette or a look–up table. Select the one you prefer and define the remaining parameters in the same way that you did when adjusting grid displaying parameters.

However, only one variable can be represented in a grid, while many variables can be associated to a vector layer, all of them stored in the corresponding database. From all of them, you must choose the one that you want to use to do the color assignment. The color ramp (if you use a color palette) will be set between the minimum and maximum values of that variable in the considered vector layer. To select the field to use, look for it in the *Color By...* list.

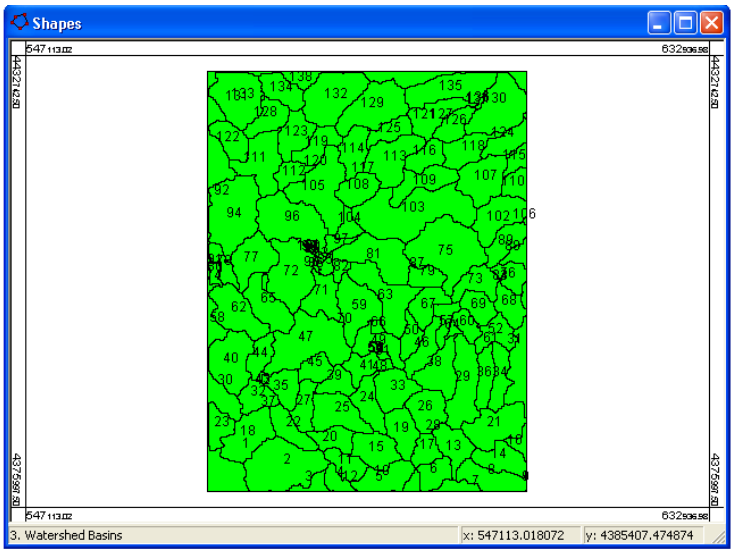
This color will be used to fill the interior of shapes, but you can put a border around them (only in the case of polygons). Set the *Border* parameter to true and select the color for the border in the corresponding *Color* field.

The interior doesn't have to be opaque. You can choose other options from the *Fill Style* list at the bottom of the window. Using the *Transparent* style will let you see both the rivers and the basins, even if you don't change the order in which they are drawn.

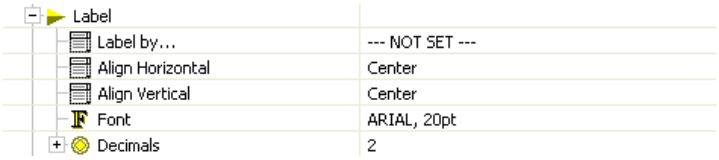
Be sure to set the *Border* parameter to true when using the *transparent* style. Otherwise you will not see anything!



While grids contain only numerical information, the database associated with a vector layer can also include text information (i.e Names of the cities represented by points or the rivers represented by lines). This information can be shown in the view by adding labels close to each vector entity, as shown in the following picture.

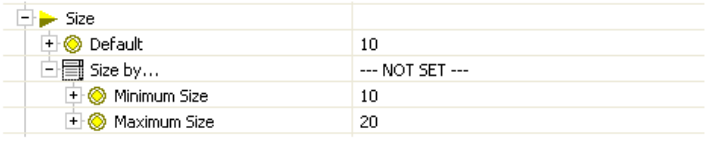


Again, you must select the field to use, this time from the *Label By...* list.



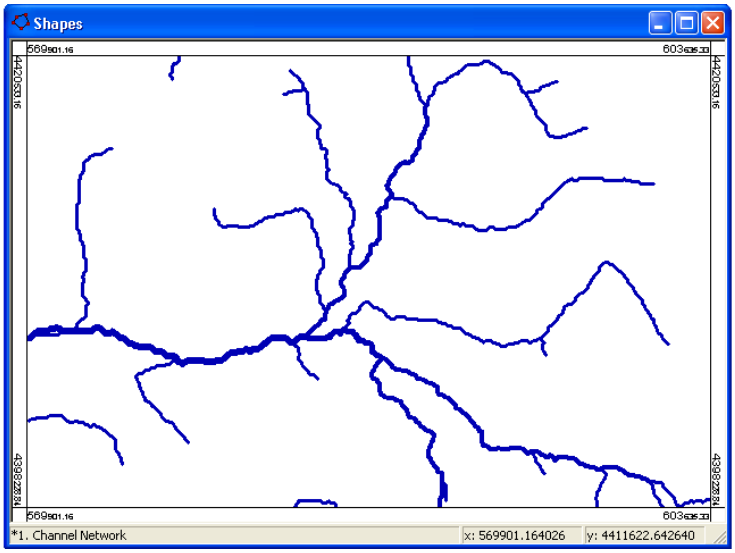
You can also use numerical fields, not just text ones. In this case, the number of decimal places to show can be set in the *Decimal Places* field.

In the case of lines and points, you will find new fields under the *Size* node to control how the shapes are drawn.



You can select a fixed width (or diameter, in the case of points), or assign a different width to each shape using the values of any field in the associated attributes table.

The following picture shows a portion of the test.shp vector layer, with line width depending on the *Order* field.



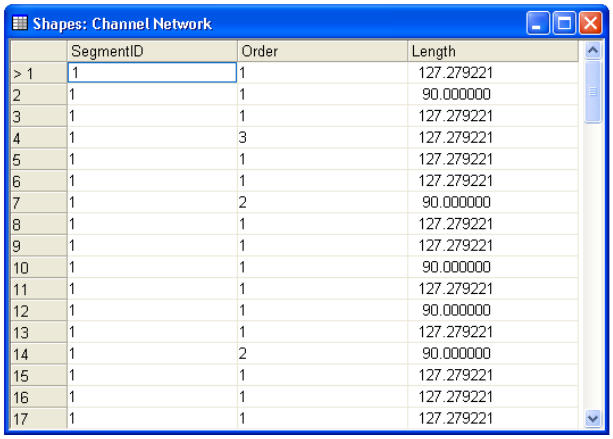
Also, when working with lines, the *Style* field will contain different styles, not related with the inside area as in the case of polygons, since lines are one–dimensional entities.

5.3 The attributes table

As it has already been stated, there is a database linked to every vector layer. You can see its content and even change it, as we will discover in this section.

To see the associated database, right–click on the name of a vector layer and select the *Attributes/Table* menu item

If you selected the test.shp layer, you will see a table like the following one.



	SegmentID	Order	Length
> 1	1	1	127.279221
2	1	1	90.000000
3	1	1	127.279221
4	1	3	127.279221
5	1	1	127.279221
6	1	1	127.279221
7	1	2	90.000000
8	1	1	127.279221
9	1	1	127.279221
10	1	1	90.000000
11	1	1	127.279221
12	1	1	90.000000
13	1	1	127.279221
14	1	2	90.000000
15	1	1	127.279221
16	1	1	127.279221
17	1	1	127.279221

Apparently, this is just a normal table, much like the ones we saw in the last chapter. However, a closer look to the toolbar reveals that some buttons are missing. Yes, you are right, you can neither add rows to the table nor remove them from it, but you can still change the values in the cells that already exist. Why is that? Well, you should not forget that this table is linked to a vector layer with spatial entities, and that each row in the table represents the data of one of those entities. If you add a new row, to which entity would it be associated? Since this entity has to be defined, the only way to add new elements is using the edition tools included in SAGA, with which we will deal in the next section.

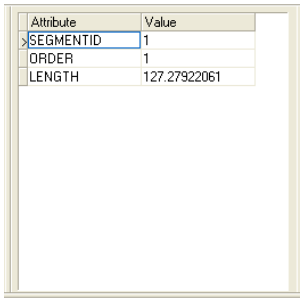
Apart from modifying the values in the table, not many more things can be done.

You can create diagrams by using the *Attributes/Diagram* menu item, and they behave exactly like the ones we saw in the previous chapter.

In the shapes toolbar you can find an info tool:

Info Tool

If you click on it, a new window will appear next to the project window. If you now click on any entity of the active vector layer, SAGA will show in this window all the information associated with that entity.



That way, you can see the values of a shape without having to look for them in the table, but getting them in a more "visual" manner.

If you want to get information about the geometrical properties instead of about the associated database (only in the case of a layer with polygons), you can right click on a shape and you will get a dialog like this.

Notice that, when you click on a row, its corresponding shape gets selected in the view. That is quite useful to locate a shape using its attributes.

5.4 Editing vector data

In case you want to edit more than the attributes table of a vector layer, SAGA offers you some simple functions that will help you edit the shapes already included in a layer, delete them or add new ones.

We will work in this section with a layer containing lines (the test.shp layer) but the methods are very similar if you are working with points or polygons instead (I invite you to try them yourself).

5.4.1 The vector hierarchy

Before we start editing a layer, is important that you understand the hierarchy under which vector entities are stored, because you will need to use it when adding or removing elements. Also, it will help you understand the terminology used in the menu items and, thus, will ease your way through this section.

On the top of this hierarchy is situated the layer itself. A layer is composed of shapes, and those can be polygons, lines or points (SAGA can handle also multipoint layers, but they are conceptually similar to points, so I will ignore them by the moment to keep this as simple as possible.) In a layer, only one type of vector data can exists. That means that points and polygons can be shown in the same view, but they have to be kept in different files.

Each shape is related to a row in the corresponding attributes table. Normally, a shape has a single part, but SAGA supports multipart shapes. To understand this, let's see an example.

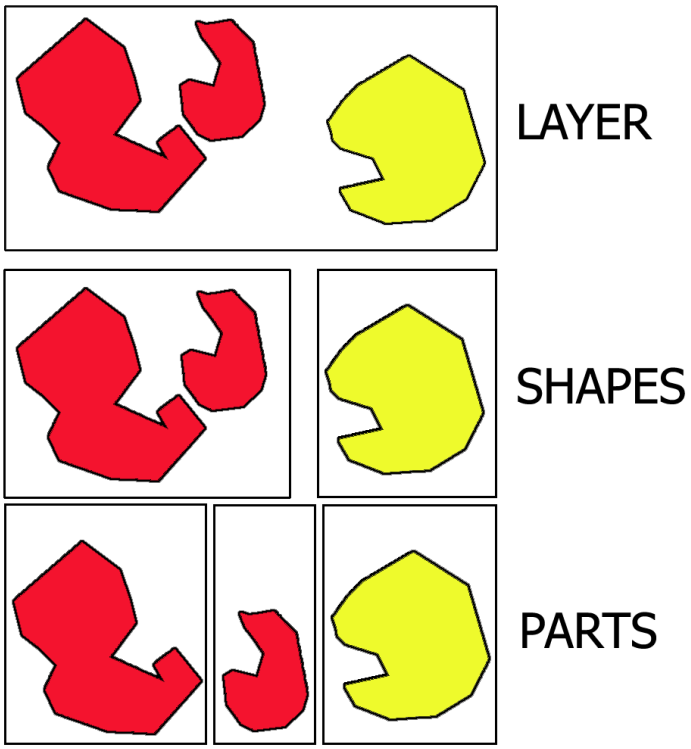
Imagine that we have a vector layer with polygons that represent countries. In its attributes table we can find fields such as *Population, Area, Gross Product* and other similar ones. While a country like Germany can be represented using a single part, other countries like Spain or Portugal cannot, since there are portions of land like islands that, although belonging to these countries, are isolated. If you can't draw all the shape without lifting the pencil from the paper, you must use severals parts to define that shape.

In other words, parts are used to break down a shape into many different elements (parts), so a single entry in the attributes table can be linked to several isolated elements that, nevertheless, share some common properties.

Finally, parts are defined by points. All the points of all the parts in a shape are responsible of the spatial properties of that shape.

According to this, all vector data can be ultimately reduced to point information. Lines are defined by its nodes (which are points), and polygons by its vertices (points as well), and when editing a vector layer (whichever type of data it is made up of), you will have to edit points directly.

I'm sure the following picture will help you understand all the above information.

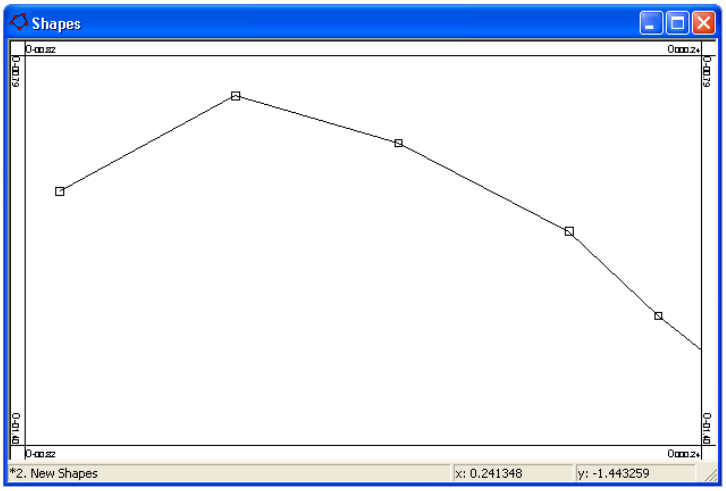


5.4.2 Editing shapes

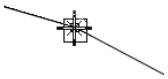
Once you understand this scheme, you can start editing the test.shp layer. In it, each segment is a shape that comprises one single part, so parts and shapes are the same here.

The first thing to do is to select the shape we want to modify. Click on the *Shapes/Display/Select* menu item (you can also use the *Select* button, represented with an arrow in the toolbar). After that, go to the shapes view window and select the shape you want to edit. Its color will change from its current color (blue if you haven't changed the shapes displaying settings) to red. Zoom to the selected shape until you see it full screen.

Once the shape is selected, click on the *Shapes/Edit/Edit Selected Shape* menu item to enter the editing mode. The points that constitute the vertices of the selected line will be represented using small boxes. That indicates that the line is ready to be edited, and those boxes make it is easier to act on the points.

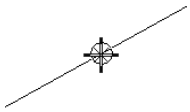


If you now move the mouse close to the selected line and its points, you will see the mouse pointer change. When it is close to a point (a little box), it will look like this:



That means that you can move the point, just by clicking and dragging. Try to move a couple of points from their original position to a new one. When you click on a point to move it, it is select and the color of the little box turns from black to red.

If you place the mouse close to a line, SAGA will give you the chance to add a new point. The more points you add, the easier it will be to give the line the shape you want, and you will be able to do it more precisely. The mouse pointer in this case will look like the one below these lines.



When you see this mouse pointer, just click and a new will point will appear. Now you can move it as explained before.

Something you can also do to edit a shape is to delete points or parts (in this case, if you delete a part, you delete the whole shape). When you click on a point and its box becomes red, the point and the part to which it belongs get selected. Now, you can use two functions to delete the point of the whole part, namely *Shapes/Edit/Delete Selected Point* and *Shapes/Edit/Delete Selected Part*. You can also find these menu items in the menu that pops up if you right–click on the the view window.

Since I suppose you have already mastered the ins and outs of the shape/part/point hierarchy, we will now try to add a new part to or vector layer. A part should be into a shape, so the part we are going to add will get included in the shape you have been editing. However, since it is a new part, it will not be connected with all the lines and nodes that you have been editing during the last paragraphs.

To create a new part, right–click on the view and select the *Add New Part* menu item. It seems like nothing has happened, doesn't it? Well, not really. If you now click on the view, a point will be added for each click you make.

The difference between the different vector data types can be seen in the way SAGA behaves depending on the data type of the layer. In a line layer like this one, after you introduce one point, when you move the mouse you will see a line between your current mouse position and the last point that was introduced. When you want to finish editing this part, you must supply a last point, and this is done by right–clicking instead of left–clicking. This resembles the way profiles are defined on a grid view.

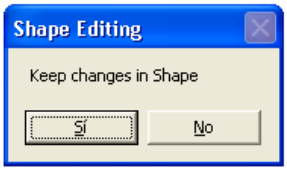
If you are working with a polygon layer, the process is quite similar, also needing a last point to close the polygon, but the lines that appear as you add points are always closed.

In the case of a points layer, the thing is quite different, since there's no need for a final point. The point you introduce is itself a final point. That means that a point part contains only one point (in fact, the whole shape contains only one point, since point layers don't support parts). If after introducing a point you click again on the view, the previous point will disappear.

Independently of the data type, once you finish a part (or it gets automatically finished in the case of points), you cannot add more points. If you click on the view, nothing will happen. You can now edit the part you have already defined or add a new one.

A multipoint layer is like a point layer but can contain several points in a shape. As you add new points, the previous ones will not disappear. The part doesn't have to be finished. Multipoint layers, however, are not used as frequently as the other layer types.

When you have edited a shape, added new parts or removed others, these changes have to be updated in the vector layer. Select the *Edit Selected Shape* menu item to unmark it and SAGA will ask you if you want to keep all the changes you have made to the shape.



5.4.3 Adding a new shape to the layer

Apart from modifying the shapes already contained in a vector layer (whether modifying its parts, deleting them or adding new ones), you can also add new shapes (and, thus, new rows in the associated attributes table).

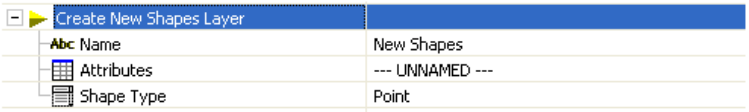
To do it, simply select the *Shapes/Edit/Add New Shape* menu item and start editing in the view window. It is like adding a new shape to an already existing layer which contains no parts at all.

Go to the corresponding attributes table to introduce the information about the just created shape.

5.4.4 Creating a vector layer

Vector layers can also be created from scratch. Once a new vector layer has been created, you can start adding shapes and adding parts to them, and also editing the information in its attributes table.

To create a new vector layer, select the *Shapes/Project/New Shapes Layer* menu item. You will see the following parameters window.



Select in the *Shape Type* field the type of data that the new layer will contain. The associated attributes table also has to be defined. Select the *Attributes* field and a table definition window will appear. Introduce the name and type of each field that will be contained in the table, in the same way that was explained in the *Creating a table* section in the preceding chapter. Close the window pressing the *Close* button.

Type a name for the new layer in the *Name* field, press the *OK* button and the layer will be created and ready to be edited.

Chapter 6

Creating layouts

6.1 Introduction

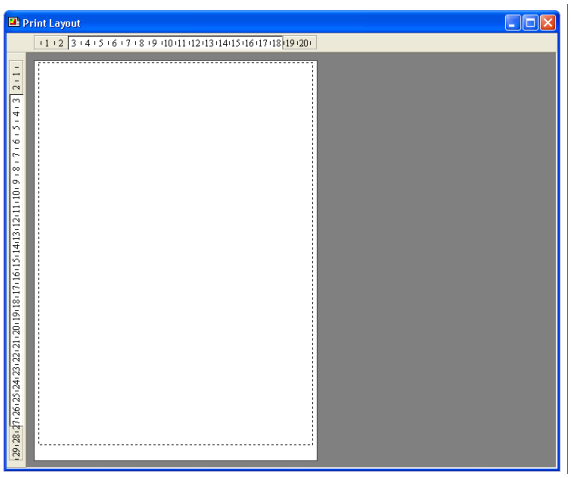
Grids (and also vector layers) look good on screen. You can select the coloring you want, shade them to get a relief effect, and do many other things with them in order to make them look nice and impressive. However, those elements usually have to be printed and included in a document, and printed information has some different needs than the one shown on screen.

SAGA can assist you formatting your layers, so you can create ready–to–print designs with a much more improved appearance than if you simply print an isolated layer.

6.2 Creating a layout

To create a print layout, select the *Print layout* menu item in the *Window* menu.

You will see a window like this.



A layout must be understood as an empty canvas where you can add elements regarding all data layers currently loaded into SAGA (therefore, to follow this chapter, make sure that you have at least one raster layer and one vector layer loaded). But before we can start adding elements, we need to define the properties of that canvas, that is, of the paper in which the layout is to be printed.

Select the *Layout/Print Setup...* menu item to see the typical Windows print setup dialog. Configure the size and orientation so they adjust to your needs, and then close the dialog. I will use a DIN A3 paper size with landscape orientation for the following examples.

Once SAGA knows the dimensions of the layout, it can add scale parameters when you add a grid or shapes layer to it.

Adding objects to a layout is pretty simple. Just select one of the menu items under the *Add Object* menu and then click and drag onto the layout in the place where you want to put the selected element. You can later resize the element you added, by clicking on its border and dragging to expand or shrink it.

Each element has its own properties, so we will study them separately.

Removing an element from the grid is also quite easy. Right–click on it and select the *Delete* menu item.

To move an element, click on it to select it and then click inside it and drag to the new desired location.

You can change the layout view by using the zoom tools, in a similar way to what you would do with a shapes view. However, two new buttons can be found in the toolbar:

Fit Width

Fit Height

When you press these buttons, SAGA adjusts the zoom automatically to the greatest scale that allows you to see the full width (or height) of the layout.

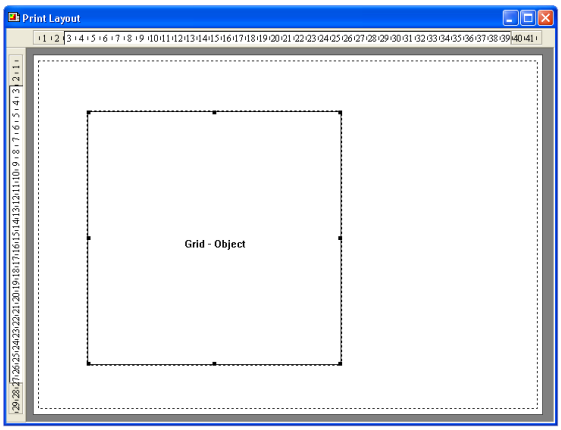
There are also three buttons with predefined zoom factors (50%, 100% and 150%), but I guess they need no explanation.

6.3 Adding grids

Since you are likely to use grids to perform analysis, you are also likely to include them in your layouts. Additionaly, the information usually found in layouts is related to output data (layout are normally created to present results), and most of SAGA modules create new grids as their output.

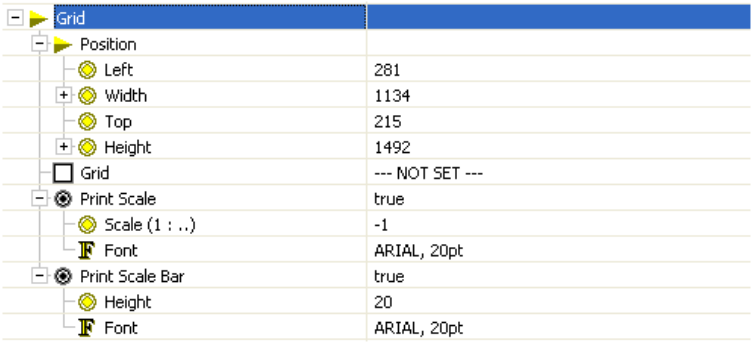
Including a grid in a layout is as simple as including any other element, just clicking and dragging onto the layout.

Once you define the bounding box inside which the grid element should be located you will see nothing but at an empty box with a label inside, similar to the one below.



That indicates that the box is suppose to contain a grid, but SAGA still doesn't know exactly which grid among all the ones than are loaded. Apart from this fundamental parameter, you can set a few more to adjust how SAGA puts the grid into your layout.

To set them, right–click on the grid and select the *Settings* menu item (You can also double–click on the grid). The typical parameter window will appear.



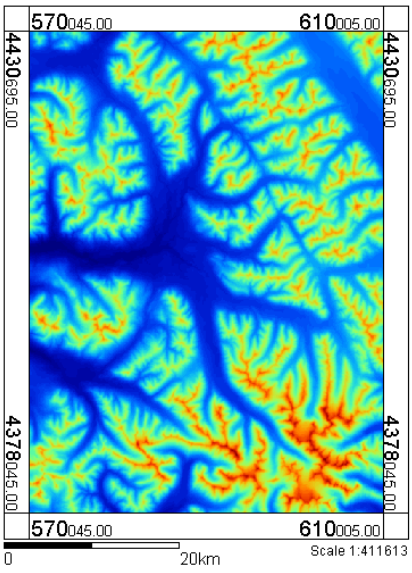
The first parameters allow you to set the position of the grid, but it is by far easier to do it using the mouse.

The most important parameter is the one labeled *Grid*, where you should select from the list the one that you want to include in the layout.

A grid can have two additional elements by it, namely a scale value and a scale bar. By default both of them are shown (their fields are set to true) buy they can be hidden by setting to false the corresponding fields.

The scale value is calculated using the size of paper that you chose in the print setup dialog and the size of the grid element. However, you will probably like to use a round number, something like 1:50000, 1:100000 or similar. To do that, introduce this number in the *scale(1: ..)* field. When you close the window, it will automatically adjust the size of the grid to fit that scale.

The grid with its related scale and scale bar will look like this:



All the characteristics of the grid that cannot be set in the parameter window (such as color palettes, classification type, or even the size of the frame in which corner coordinates are written) are taken from the own settings of the select grid. If you modify them, SAGA will automatically update the layout view with the new settings.

Apart from modifying the look of the grid object using grid settings, you can change the area shown in the layout by using the zoom tools in the grid view window. The grid element

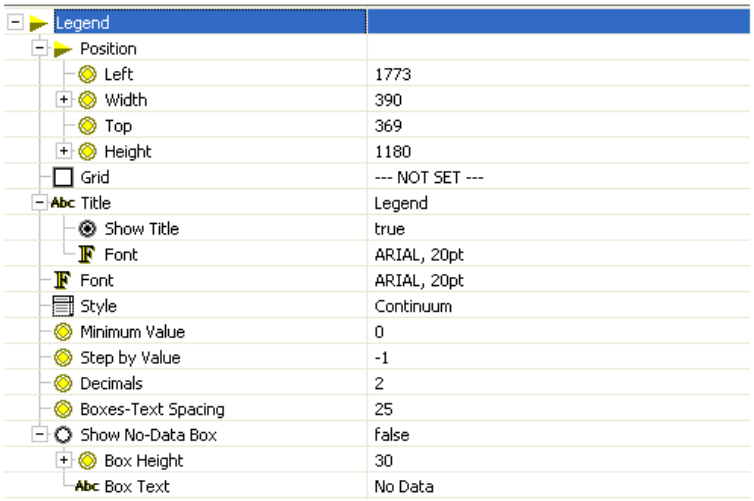
in the layout is linked to the grid view, so it will show just the area within the current extent of the grid in its own grid window. Since the scale doesn't change unless your resize the grid object or modify it in the settings window, the object might change its dimension to adjust itself to that scale.

6.4 Adding grid leyends

Since you have adjusted it in the grid settings, you know the meaning of the colors used to display a grid. Also, you can move your mouse over it and know the exact value of cells. However, once the grid is printed, this information is not available, and the final user of the printed layout will probably not know what all those colors mean. A legend is something you must include to "explain" the information contained in the layout.

Select the *Layout/Add Object/Grid: Leyend* menu item and then click and drag onto the layout to define the area where the leyend will be placed.

Again, you will not see a legend but an empty box, since you have to adjust its properties and link it to a grid. Double click on it to get to its settings window.

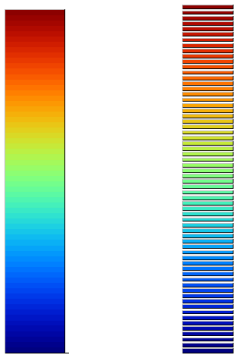


Along with the parameters that define the position of the legend in the layout, you will find the *Grid* field in which you have to select the grid used to create the legend. This legend will show the colors used in the grid view, and the values that are represented by each color.

If you want to add a title to the legend, set the *Show Title* field to true and introduce the title text in the *Title* one. The font characteristics can be defined clicking on the *Font* cell under the *Title* node.

The font used for the legend values, can be set using the remaining *Font* field.

Legends for continuous grid are not the same as those used form discrete grids. Therefore, SAGA offers two types of legends, as it is shown next.



To select one of these types, use the *Style* field. If you choose the *Continuum* style, you will get a legend like the one on the left, while choosing the *Boxes* style will give you a legend like the one on the right.

No data values are not included in this scale, which only contains those values between the min and max values of the grid. To show a box containing the no data value color, set the *Show No–Data Box* to true and define the dimensions and text of this box using the *Box Height* and *Box Text* fields.

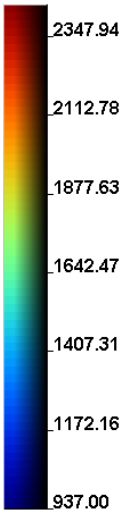
The appearance of a continuum legend can be adjusted by changing the values in the fields below the *Style* box.

Modify the *Step by Value* field to change the number of text labels containing values that will appear on the right side of the color ramp.

The *Box–Text Spacing* field contains the horizontal distance between the color ramp and the text labels.

The number of decimal places used in the text labels can be set modifying the *Decimal Places* field.

If the grid that is related to a legend is shaded, the legend will also have a shaded effect, so as to reflect the different color tones that can be found for a same value.

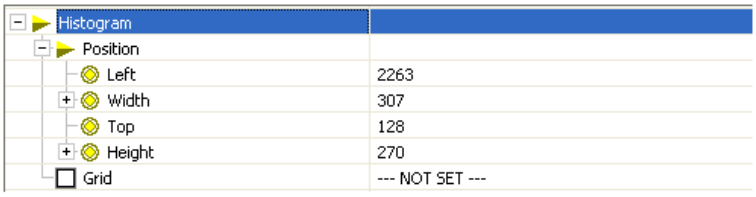


6.5 Adding a histogram

Adding a histogram is easier than adding a grid or a legend, since no parameters must be set at all (except for its position in the layout and the grid from which the frequency data has

to be taken). Select the *Layout/Add Object/Grid: Histogram* menu item and then click and drag onto the layout to define its extension.

Select the grid you want to use in the *Grid* field of the parameters window.



Histograms in the layout are linked to "real" histograms, so a histogram windows must be open. Otherwise, you will just see a blank box, as if no grid had been linked to the histogram element.

6.6 Adding a profile

If you know how to add a histogram to a layout, you already know how to add a profile. Select the *Layout/Add Object/Grid: Histogram* menu item, define its extension in the layout and link it to a grid. As in the case of histograms, a "real" profile must be created, so go to the grid view window and create one using the grid that you linked to the layout profile.

As you change this profile, it will also change automatically in the layout.

6.7 Adding 3D Views

The last element related with grids that you can add to a layout is a 3D view. Again, adding it is as simple as defining its extension and linking it to a grid. The characteristics of the 3D view are controlled using the own 3D View parameters window, which means that, as in the preceding cases, a 3D window must be open or the 3D View element will otherwise appear as a blank box in the layout.

6.8 Adding shapes

Shapes are not different from grids, except for those things that we already saw in the *Working with shapes chapter*.

Therefore, parameters such as position and dimension, or the presence or absence or scale bars, can be set in the corresponding parameter window, which resembles the one previously used with grid objects. The only difference is that you don't have to select a grid, since all shape layers are represented simultaneously (that's why, when you add a shapes object, you will not get an empty box, but already an image of the current shapes view).

To add a shapes element, select the *Layout/Add Object/Shapes* menu item and draw its bounding box in the layout.

As it happened with grid objects, shapes objects are also linked to their corresponding view window, considering this time not only its extent, but also its drawing order and other settings that were not found in the case of grids. Therefore, to control which vector layers are included in the layout, simply hide the ones you want to leave out in the shapes view window.

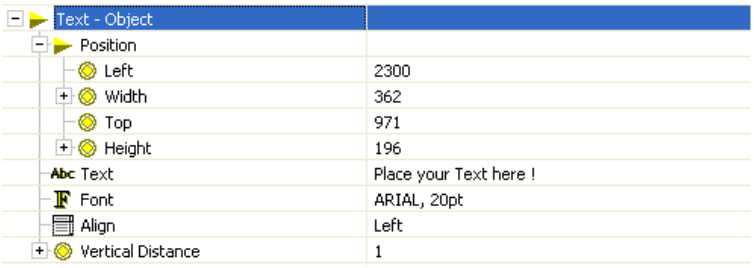
In short, shapes objects behave quite similar to grid object, so you will probably not have any problem with them at all.

6.9 Adding text

Grids and shapes are the most important elements of a layout, but the appearance of a layout is quite poor if no additional text is added. Some elements such as legends or grids have their own text labels, but they are not enough to give a full description of what the layout represents. Whether you need to put a title in the layout or an explanation under a grid element, the text capabilities of SAGA will prove highly useful.

To add a text box, select the *Layout/Add Object/Text* menu item and define its extension clicking and dragging onto the bounding box. You will see inside a box a small text (the default font size is so small that, in fact, you might not be able to read this text if the zoom is set as full extent)

Double click on the text box to edit its content.



To edit the text, click on the *Text* field and you will get to a text editing window.

Simply write here the text you want to appear in the text box. You can change the font that is used in this window using the *Font* button, but this will have no effect on the text that will be print on the layout. To change the characteristics of the text in the layout, close the text editing window and use the *Font* field of the parameters window instead.

The way text is aligned within the bounding box that you defined can be selected in the *Align* field.

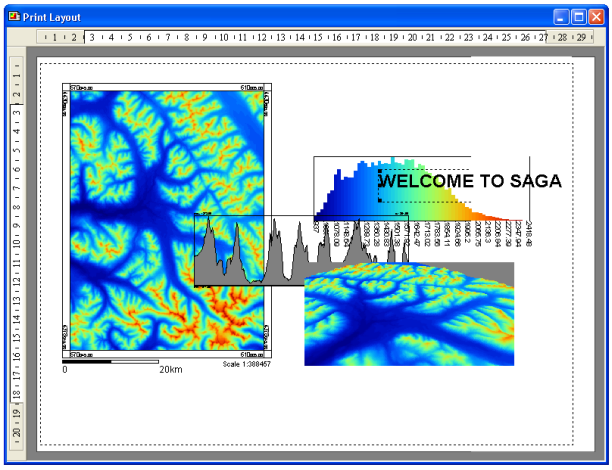
Lastly, if your text contains several different lines, the distance between these lines can be changed modifying the value in the *Vertical Distance* field.

6.10 Arranging layout elements

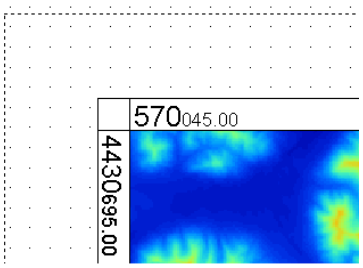
Now you know how to add the different elements to the layout and adjust their properties. However, there are still some adjustment that can be made and will have their effect in case some of those elements overlap.

If you remember from the *Working with shapes* section, if several layers overlapped, you should set the drawing order so as to avoid some information to get covered and disappear. The same happens with layout elements. If you right–click on any of them, you will find four menu items (with the same name as the ones in the *Grid* menu) that can help you set the right order for your elements.

The following figure shows a layout(not a very stylish one, I must admit. . .) where elements overlap showing the meaning of the drawing order. As you can see, blank areas in profiles and histograms behave like transparent ones, so you can see through them independently of their position in the drawing order.



When arranging your elements in a layout, its a good idea to keep them aligned to get a cute appearance instead of a rather messy one. While you can freely move each element until you get a good alignment, there's a more precise way to do it. Select the *Layout/Align objects to grid* menu item and a dotted grid will appear.



Now, whenever you move an object, it cannot end in any position within the grid, but its upper–left corner must be located exactly in one of the dots of the aforementioned grid. That way, it is easier for you to know when two object are correctly aligned.

Sometimes elements of a grid are tightly related, so tightly that you would even like to join them so they can be considered as a single one. To do this SAGA has a *Group* command that groups all the element that are select. To select more than one single element, use the *shift* key and click on all the ones you want to select, while holding it.

Elements that are grouped move as a single one, so if you grouped them when they were aligned, they will keep their alignment. To revert to the previous configuration, right click on the group of elements and select the *Un–group* menu item.

6.11 Once the layout is finished

After you finish designing your layout, you will like to do something with it. The most usual thing you will do is printing it, but other things can be done with it that will help you creating other layouts later or maybe improving it in another software.

If you simply want to print your layout, just go to the *Layout/Print* menu item. You can see a print preview using the *Print preview...* menu item. These are the typical functions found in almost every Windows application with printing capabilities, so there is no need for further information.

If you plan to use your layout in an image processing software, you will need to save it in a compatible format. SAGA supports the Windows metafile format, and you can save your layout using the *Layout/Save as Window Metafile...* menu item. No other format is supported, but, once again, there are many applications for converting between graphics file formats (and a lot of them are free ones).

Also, you can select objects in the layout and copy them to clipboard using the *Copy Selected Objects into Clipboard* menu item, although this is a rather confusing option. Don't forget that, eventually, you will print the layout or the modified version of it you create with the objects that you copied into the clipboard. Scale here is a very important thing to consider, and if you resize those objects in another software, the scale values will not be affected, thus showing a wrong scale. Always keep this in mind when moving elements between SAGA and any other software.

If you have created a template, you can later use its structure as a template. This is useful if you have to include your layouts in a project (all the layouts should have a similar appearance) or simply if you really like your design and don't want to spend your time designing it again the next time.

To save a layout, use the *Save Layout...* menu item. To open a layout, use the *Load Layout...* menu item. When you load a previously saved layout, you will have to adjust some properties of its elements (remember, double click on each of them and you will get to the parameter window), but the design of the layout and its most fundamental characteristics (the ones that usually take longer to adjust) will not need any adjustment at all.

Chapter 7

A first step into modules. Some basic grid modules

7.1 Introduction

Now you know how to handle all of SAGA basic functionalities. You should feel comfortable in the SAGA environment, and, if you have carefully read the preceding chapters, you should be prepared to deal with any kind of new result that SAGA might create, whether a grid, a vector layer, a table or whatever.

It is time, then, to start studying some of the most important SAGA modules and explore their possibilities.

This chapter describe the modules implemented in the Grid_Tools.mlb and Grid_Calculus.mlb module libraries. Therefore, be sure that these libraries are already loaded.

The modules in these libraries are mostly used to prepare grid layers and modify them, so they can be ready for being used as source data in other modules or for any other purpose. Things such as filtering a grid or modifying its cell size, among others, can be accomplished using this modules.

Except for the *Grid Calculator* one, most of the modules described here don't offer results in a strict sense, but their importance is, nevertheless, enormous. You will surely discover it once you really start working with grid data for real purposes (the real world is always tougher...)

7.2 Filtering a grid

Let's start our trip into the vast realm of modules with an easy group of them: the filtering modules. You can found them in the *Modules/Grid/Calculus/Filter* menu (since all the menu items regarding modules are under the *Modules* menu, I will not write it from now on to avoid large menu paths. In this chapter, all of them are under the *Grid* menu, so sometimes I might omit it).

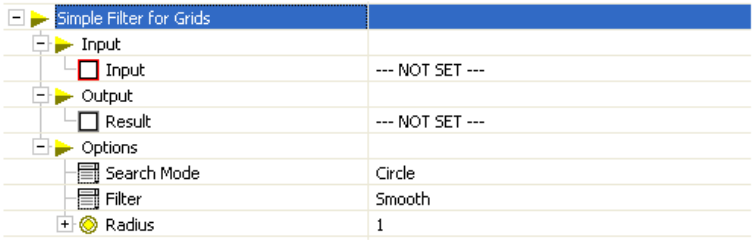
First of all, what is a filter? A filter modifies all the cells in a grid (applying different algorithms and formulas) to generate a new version of that grid. The grid you obtain after a filtering process represents the same variable and in the same units as the original grid. Users acquainted with digital image processing will have no problems understanding this concepts.

The most common use for a filter is the elimination of noise. Single cells with unexpectedly high values are a good example of what *noise* means. When dealing with images instead of data grids, filters are a basic tool that is constantly used to prepare those images before any further processing. However, you should be careful when applying filters to data grids, since

their effect can significantly alter their information. For example, applying a smoothing filter to a DEM is a somehow "rude" way of eliminating small sinks, but is not recommended, since it changes the height information contained in all of the cells, while only some of them should be modified. Although it removes noise, a smoothing filter causes the filtered grid to have a lower lever of detail than the original one.

Always have this in mind when you use a filter: The less you modify the grid (assuming its information is reliable and comes from a good source), the more accurate your results will be.

To get a feel of what filtering means, let's filter our already familiar test.dgm grid. Select the *Filter/Simple Filter for Grids* menu item to get to the following parameters window.



I will use this first module to introduce some of the features that are common to most modules, so once you learn them here there will be no need of explaining them again each time they appear (thus saving me a great effort and making the text more fluid and less boring for you)

The first thing to do here is to select the grid that will be used as input. You will find this in each and everyone of the SAGA modules, since all of them need some input to work with. This input, however, will not always be a grid, but also a vector layer or even a set of several different input elements.

Select the test.dgm grid as input. You can see that there is a red box in the *Input* field, as opposed to a black one in the *Output* one. That means that this input is compulsory. Here, only one element is needed as input, but we will see other modules where more than one element can be used as input, some of them being optional.

Once the input is set, you have to select the output. Again, you will see this in all modules, for always some kind of output is generated. Whether this output is a grid, a shapes layer or table, you will always find the same elements in the corresponding output field (or fields): a — *NOT SET* — , a — *CREATE NEW* — value and then the names of all the elements that are currently loaded in SAGA and whose data type is the same as the data type of the output (grid, shape, etc.). If the result is a grid, only those grids included in the current project will be shown.

I will explain what all this options mean.

- — **NOT SET** —: If an output field is set to — *NOT SET* —, SAGA will not generate that result. There is no point on using this in here, but some modules generate several elements as result and one might only be interested in some of them.
- — **CREATE NEW** — : Using this option will cause SAGA to generate a new element from scratch. That is, if you have a project with just one grid and select this option, your project will have two grids once the module has finished its work.
- **Name of element**: Selecting the name of an already existing element will cause SAGA to put the results of module execution in the selected element, overwriting it. Don't forget that SAGA has no undo capabilities, so if you overwrite a grid you will not be able to recover it.

In this case, selecting the test.dgm grid as output is an interesting option, since you might want to substitute the original grid with the filtered one. However, select — *CREATE NEW* —, so you can later see the difference between both grids.

The way the simple filter is performed can be adjusted by modifying the values in the *Search Mode*, *Filter* and *Radius* Grid. First, you should select the type of filter you want to pass to your input grid. These are the available filters:

- **Smooth**: Smoothing a filter will soften the difference between a cell and its surrounding cells. The 2D representation of the grid will get blurred. The 3D view will show a smoother relief. The new value is calculated using the following formula:

$$z' = \overline{z} \tag{7.1}$$

  Where $\overline{z}$ is the average value in the analysis window.

- **Sharp**: This filter has the opposite effect to the smoothing one, accentuating the differences between cells. Its formula is as follows:

$$z' = 2 * z - \overline{z} \tag{7.2}$$

- **Edge**: Use this to detect edges and areas of high variation in a grid. The resulting is not a grid similar to the input one, but rather different, so you will not be able to use it just like the original one. The formula for this module is:

$$z' = z - \overline{z} \tag{7.3}$$

Since the new values assigned to each cell are calculated from the original values of the cell and the values of its surrounding ones, the number of cells to consider around the central one has to be defined. To do that, you have two parameters, namely *Search Mode* and *Radius*.

Using the *Search Mode* you can select a round or square group of cells to be considered around the central one. The size of this group, expressed as the number of cells to consider to each size of the central cell, can be set using the *Radius* field. The higher the value you introduce, the more accentuated the effect of the filter will be.

After all the parameters have been set, press the *OK* button. You will see how SAGA works, and its progress will be shown in the progress bar on the lower–right part of the main window. Once it has finished filtering the grid, you will have a new one in the grid project window.

You have just used your first module and performed your first processing operation on a grid. As you can see, using this module was not really hard, just a couple of parameters to set and not much more. Notice how, after you use a module, a new menu item appears in the *Modules* menu. The last four modules that have been used have its own menu item here, so they can be more easily used next time.

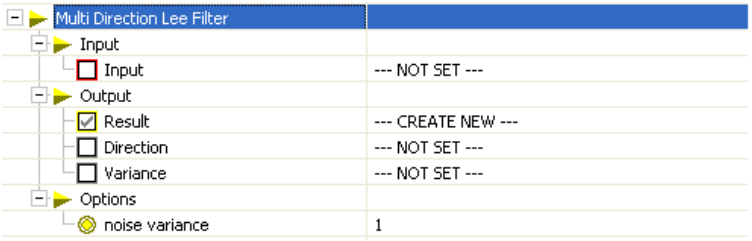
The usage of the other filter modules is quite similar, so instead of explaining how to use them (which would be redundant) I will describe the effect that each filter has on the input grid, and when you should use each one of them.

The second filter menu item you can found is the *Gauss Filter for Grid*. A Gaussian filter is an smoothing filter, but its mathematical background its a bit more complex than the simple filter. Also, its smoothing effect is "stronger" then the one you can get with the simple filter.

To adjust the smoothing intensity, use the *Standard Deviation* parameter. You will find as well the usual *Radius* and *Search Mode* fields. The greater the *Standard deviation* value, the greater *Radius* you should introduce.

The next menu item you can find is *Laplacian Filter for Grids*. Unlike the Gaussian filter, a Laplacian filter is an edge detection filter. It is a commonly used filter in image processing, but a not so usual one with other kinds of grids. Passing a Laplacian filter over a DEM can be used to the study landforms or detecting concave and convex areas, but we will see more detailed ways to do this once we reach the terrain analysis chapter.

The last filter available in SAGA is the *Multi Direction Lee Filter*, another noise removing filter. Thanks to its characteristics, it can be used on DEMs with lower loss of detail, since it preserves slopes. The parameters window of this filter is a a bit different to the ones we have already seen.



Here you can find multiple outputs, although by default just one of them is created. The two remaining ones are a grid with the minimum variance of each cell, and another one with the direction of the filter with that minimum variance. These directions are numbered from 1 to 16.

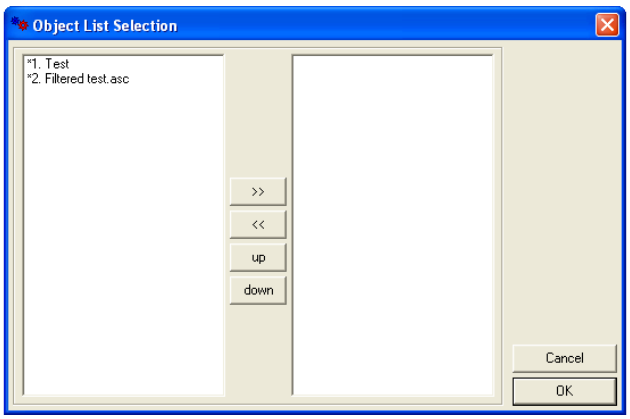
You should try all the filters and experience with them. Get some new data and apply those filters to it to see how each one of them behaves. Better yet, wait until we explain other modules and use the new grids obtained from them to test the behavior of each filter.

7.3 Arithmetic mean of grids

The grid calculator, which will be explained in this same chapter, can be used to perform mathematical operations involving one or several grids. The calculation of an arithmetical mean can, therefore, be accomplished using the grid calculator module, but there's a simpler module that we can use for this purpose, and that will help us to continue discovering how modules work, at the right pace.

Select the *Calculus/Arithmetic Mean of Grids* menu item.

As you probably expected, the parameters window associated to this module is quite simple. The only information you need to enter is the name of all the grids that you want to use to calculate their mean value. As output, only one grid is generated, containing that mean value. Since a multiple selection is required, SAGA will display a multiple selection dialog when you click on the *Grids* field.



Move to the box on the right all the grids you want to use and then select *OK* Then select *OK* in the parameters window and you will have your new grid calculated.

7.4 Normalizing a grid

Sometimes, you might need the values of a grid to be expressed using a different scale, usually from 0 to 1. This is quite common when you deal with several grids and you want to use a model in which each parameter should be scaled according to a particular scale such as the aforementioned 0–1 one. Again, you can normalize a grid simply using the grid calculator, applying this formula:

$$x'_{i,j} = \frac{x_{i,j} - \min}{\max} \tag{7.4}$$

where min and max are, respectively, the minimum and maximum values found in the grid.

However, SAGA has a single module that simplifies this task, and it will be described next.

Select the *Normalize grid* menu item. The parameters window you will see is very simple. Select an input and an output grid, and then choose from one of the two available methods.

- $(0.0 < x < 1.0)$: If you select this option, SAGA will use equation 7.4 to normalize the grid
- **Standard Deviation**: If this method is selected, the standard deviation of the resulting normalized grid will equal 1.

7.5 The Grid Calculator

One of the most useful modules that you can find in SAGA is the grid calculator. Since it is a very flexible module, it allows your to perform all kind of arithmetical operations with grids, and can be, therefore, used for a large variety of purposes.

The Grid calculator can be invoked using the *Calculus/Grid Calculator* menu item.

Its parameters window is quite simple, although it is a bit more tricky than what we have already seen for other modules (don't worry, it is still quite easy). The first thing to do is to select all the grids that we are going to use in your calculations. This can be done by clicking on the *Input Grids* field and then selecting them in the multiple selection window that will appear.

The order in which selected grids appear in the box on the right is very important, so pay close attention to it. Later, when you introduce the formula that defines the operation you want to perform, you will not call grid by their names, but using single characters that are assigned to each grid depending on its position in the selection list.

The scheme used to assign this single characters is rather simple: the first grid will be called `a`, the second grid will be `b`, the third one `c`, and so on. The grids are named in alphabetical order, so you don't have to use their full names when referring to them. The drawback of this is that you will have to remember the order in which you selected them. Otherwise, you are likely to mess the results....

Of course, after selecting the input grids, you have to select an output one. You can use a grid as input and also as output, there is no problem with that. The internal architecture of SAGA allows to do this.

Once input and output grids are set, it is time to deal with the key element of the module: the *Formula* field. Here you must write the arithmetical formula you want to use, and you have to do it according to some rules. I will try to explain all the important concepts that are

needed to get the most out of this module. You can also find some of this information in the info box of the parameters window.

Starting from the most basic concepts, you can introduce integer and real numbers and make elemental operations with them. There's no need of using grids (although this is not really useful). For example, the formula 1+5*6 will yield as result a constant grid with a Z value of 31.

Operator precedence is as usual: + and - evaluated before * and /, and those before the power operator.:

For ArcView users: ArcView Map Calculator doesn't use this same order, so if you have used it you might get a bit confused. To avoid mistakes, use parentheses whenever you are not completely sure of how elements in a formula will get evaluated.

You have to check the integrity of the formula yourself. SAGA won't complain of a formula like 4*+2/, but surely will not give a good result.

Some weird results can be obtained as well by using a character not assigned to a grid (because there are not enough of them to reach that character).

Grids are used just like numbers, you simply type in their names in the formula and that's it. For example, the following formula will calculate the arithmetical mean of the two first selected grids: (a+b)/2

The following operators are supported by SAGA:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Power $\hat{(}$)

Along with this, there are some function that you can use. Some of them just need one parameter, which you should enclose in parentheses, but some of them take two.

SAGA supports the following functions. An example is given for some of them to help you understand how they work.

- **ln(x)**: returns natural logarithm of x. (but ln(e) will not return 1 !!!)
- **sin(x)**: returns the sine of x. x must be in radians
- **cos(x)**: returns the cosine of x. x must be in radians
- **cos(x)**: returns the tangente of x. x must be in radians
- **asin(x)**: returns the arcsine of x, in radians
- **acos(x)**: returns the arccosine of x, in radians
- **atan(x)**: returns the arctangent of x, in radians
- **atan2(x,y)**: returns the arctangent y/x, in radians
- **abs(x)**: return the absolute value of x. abs(-5)=5
- **int(x)**: returns the integer part of x. int(5.4)=5

- **mod(x,y)**: returns the modulus of x/y. mod(7,4)=3

You can also perform some boolean operations with the grid calculator. False is coded with a zero value and true with a 1 value.

These are the boolean functions implemented in SAGA:

- **gt(x,y)**: true if x is greater than y
- **lt(x,y)**: true if x is lower than y
- **eq(x,y)**: true if x equals y. When using this function SAGA evaluates it in a per–cell basis. Therefore, eq(a,b) will not return 1 if grid a equals grid b. It will return 1 for those cells that have the same value in both grids, and zero otherwise.
- **ifelse(condition, x, y)** returns x if condition evaluates to true (condition=1) or y if it evaluates to false

You can combine this functions and nest them to make complex formulas, but sometimes it is better to perform several different calculations rather than trying to get the final result all at once, since the defined formula can get quite unmanageable.

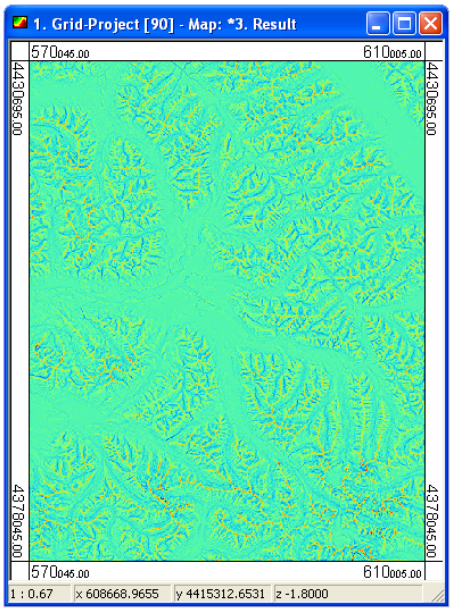
A little tip: take care with no data values. Also, if the grid calculator has to evaluate an expression such as $\frac{0}{0}$, it will return a no data value. Divisions by zero such as $\frac{4}{0}$ will not return a no data value, but an infinite value. Pay also attention to zero values when using the grid calculator.

To see how powerful this module can be, let's work out a couple of examples.

For the first example you will need an original grid and a filtered one (if you don't have it, create it using any of the filtering modules). We will try to see how the filtering process has altered the grid.

Go to the grid calculator and select both grids in the *Grids* field. Put the original one first, so it will be referred using character a. Now, type the following formula in the corresponding field: a-b.

Press the *OK* button and you will get a grid like this:



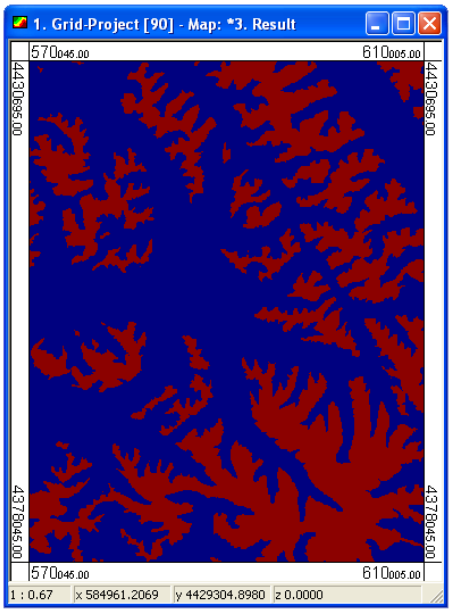
This grid represents the difference between the original and the filtered DEM. Go to the filtered one and have a look at its parameters window. Remember that it contained some statistical values. The average value of this grid will be a good indicator of how much our DEM has changed after the filtering process. The frequency histogram is also an interesting tool for this purposes.

Now for the second example, imagine that you want to create a very easy land–use grid. To create it, you will assign the value 1 to all the cells that are under 1600 meters and 0 to the remaining ones, since you consider that no vegetation grows above 1600 meters. How would you do it?

It is quite easy. Go to the grid calculator and select the test.dgm grid as the only grid to be used for this calculation. You will see that, if you have previously selected some grids for using them in the grid calculator, next time you use the module they will be already selected. You will probably see *2 list item(s)* in the *Grids* field.

Now go to the *Formula* field and enter this: ifelse(lt(1600,a),1,0)

You will get the following grid:



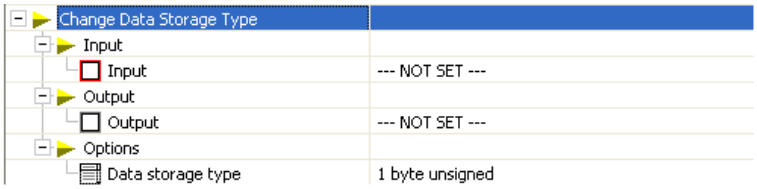
Don't worry if you don't understand the above formula. Try all the functions individually and once you understand them, try to combine them and make more complex expressions.

7.6 Changing the data storage type

You should already be aware that grids usually occupy very large amounts of memory. Until now, we have used grids containing continuous and discrete values, and nothing has been said about the precision needed to handle and store them. While a DEM must have a great accuracy, and it should be possible to use Z values such as 431.65, some more simple grids (think about the one just with 1's and 0's that we created using the grid calculator) don't need so much precision.

Higher precision involves larger amounts of memory. Therefore, it is a good idea to reduce the precision of those grids that do not need it.

To do that, we can use the *Tools/Change Data Storage Type* module. Select it and you will see the following parameters window:



Apart from the *Input* and Output fields, you find one labeled *Data Storage Type*, with a list of different storage types. As a rule of thumb, the less bytes the data storage type uses, the simpler the grid has to be to avoid data losses. The floating point storage types allow the storage of real values, while the remaining ones can only handle integer values.

Once again, if you don't understand what this means, you should not use it. Misusing this module can be a great source of problems. For example, using a non–floating–point type its not compatible with normalizing a grid, since all the values will be converted to real values ranging from 0 to 1. Since that storage type cannot handle real values, all values less than 1 will get truncated to 0.

Anyway, here is a brief description of all this data types:

- **1 byte signed**: Integer values from -128 to 127
- **1 byte unsigned**: Integer values from 0 to 255
- **2 bytes signed**: Integer values from -32768 to 32767
- **2 bytes unsigned**: Integer values from 0 to 65535
- **4 bytes signed**: Integer values from 2147483648 to 2147483647
- **4 bytes unsigned**: Integer values from 0 to 4294967295
- **4 bytes floating point**: Real values with seven digits precision.
- **8 bytes floating point**: Real values with fifteen digits precision.

In some cases, however, you might want to use integer data storage types not for memory handling purposes, but for other reasons. When dealing with discrete parameters, integer grids are usually more interesting, since Z values are usually just used for classification purposes and they do not really contain a real (not in the sense of $\mathbb{R}$) value.

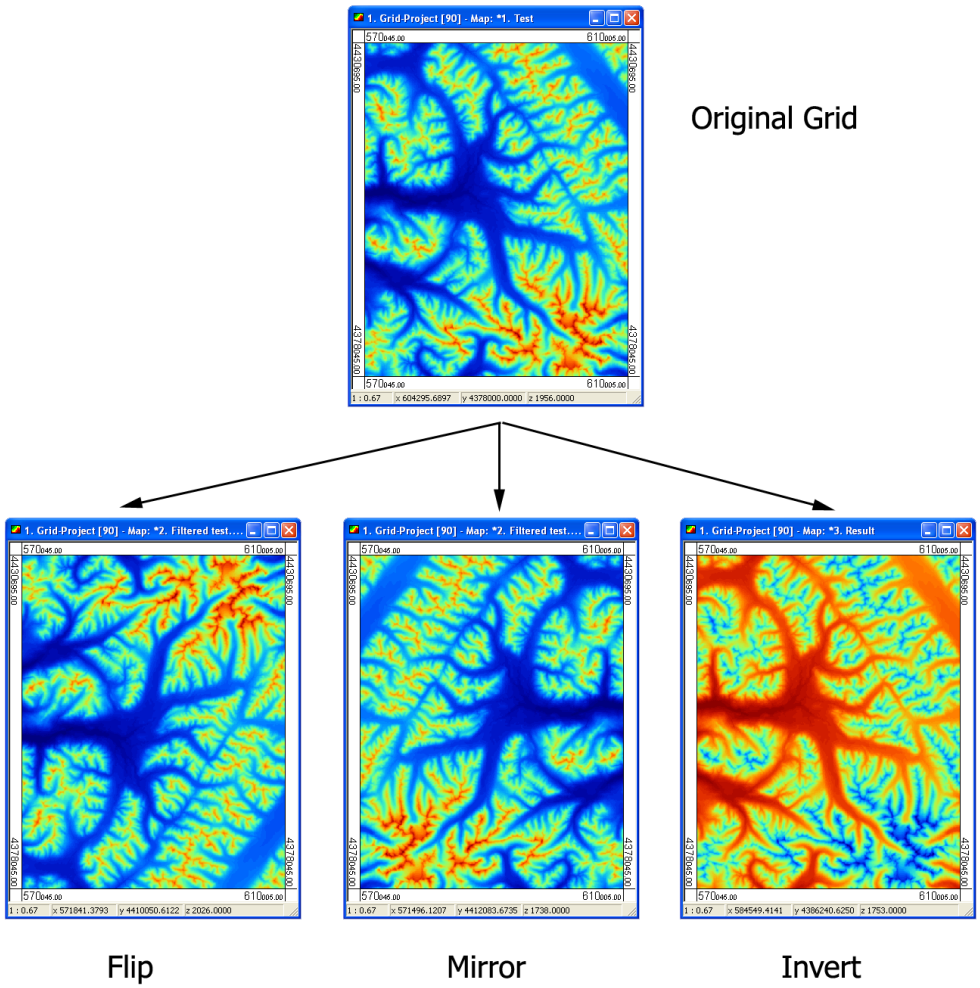
My advise is to be cautious with this module, specially when using the 1 and 2 bytes data types. Don't use them unless you know exactly what you are doing.

7.7 Changing grid orientation

Rotating or mirroring an image is something that you can find in every image processing program. SAGA also has its own module to do this basic operations.

Select the *Tools/Change Grid Orientation* menu item. You will find in the parameters window the usual *Input* and *Output* fields, and one labeled *Method*. There are four available methods, namely *Copy, Flip, Mirror* and *Invert*.

The *Copy* method will make a new grid with an exact copy of the input one, *cloning* it. For the remaining ones, I believe a picture is worth a thousand words, so have a look at the following one to understand what they do.



Maybe the *Invert* method its the hardest one to figure out. Instead of modifying the position of each cell like the other methods, it modifies the Z values, using the following formula:

$$z' = \max - z + \min \tag{7.5}$$

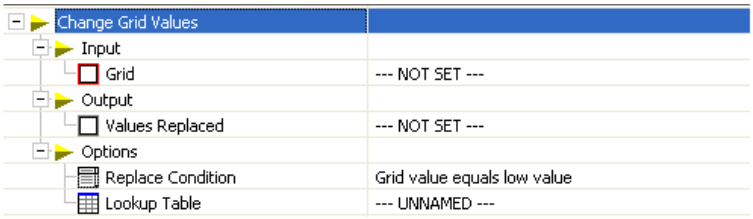
where max and min are, respectively the maximum and minimum Z values of the grid.

7.8 Changing grid values

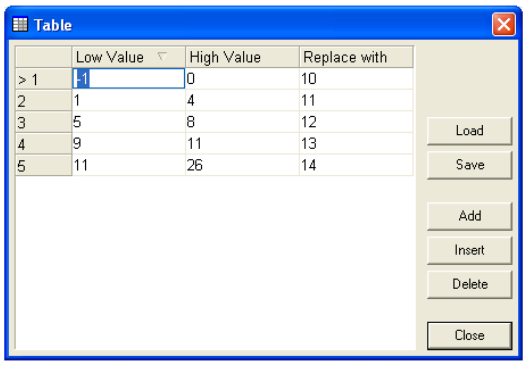
Changing grid values can be useful in some situation. If your remember the *Working with grids* chapter, we used a lookup table to define the coloring of a DEM according to its suitability for an activity (which was evaluated using the slope). Although visually the DEM would be divided into classes (good, not so good and unsuitable), actually the grid retains its values and we couldn't use in any calculation that needed the values to be divided into the three aforementioned groups. We need to change the cell values, not just their appearance.

To do this, a similar lookup table should be used, but this time in the *Change Grid Values* module instead of in the settings of the grid.

Select the *Tools/Change Grid Values* menu item to get to the following parameters window.



Clicking on the *Lookup Table* field will cause a table window to appear.



Here you cannot find a color selection cell, but the same boundary value cells that appeared in the grid settings can be found. As with other tables, you can add and remove rows and edit the cells just entering the new values you want to use.

In the *Replace with* field you can introduce the new values to assigns to the cells within the limits defined by the other two fields of each row.

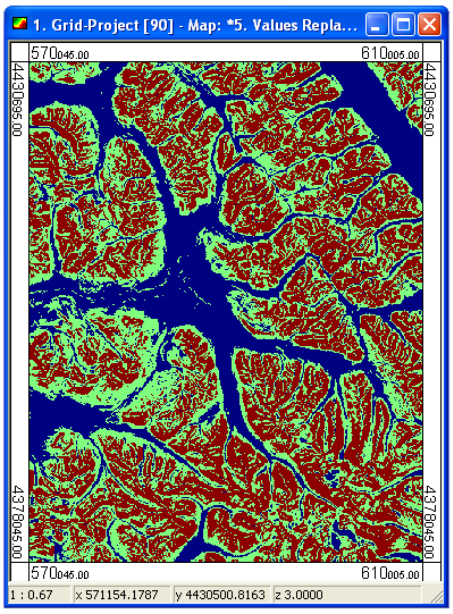
By default, only the lower value is used. All cells containing the same Z values as the one introduced in the *Low Value* field will be assigned the corresponding *Replace with* value. But this is not the kind of method we need to change the slope values into suitability ones. However, SAGA offers other alternatives. Here they are:

- **Grid value equals low value**: The default one. This is not very useful with grids representing continuous variables, but it is a used quite often with grid containing discrete ones. Imagine that you have a land–use grid divided into 10 different classes. Now imagine that land–use has changed in this area and all of the cells of one class have to be converted to another one. Introduce the value assigned to the old class in the *Low Value* field and the one assigned to the new class in the *Replace with* field, and you will have it.

- **Low value** $<$ **Grid value** $<$ **High value**: The one that we have to use to create the suitability map. All the cells whose values are greater than the lower value and lower that the higher one will get the value in the *Replace with* field.

- **Low value** $<=$ **Grid value** $<$ **High value**: Surely you will be able to figure it out yourself.

Now, for a practical result, here you have the suitability map. You can see the grid settings window to see how the maximum and minimum values of the grid have changed.



7.9 Resampling a grid

We have seen that grids that represent the same area and have the same cell size are put into the same grid project when opened or created. Let's say that you have a DEM and a land cover grid, but the latter has a cell size twice the one of the former. If you open both grids, they will be put into separate projects and you will be unable to use them together (for example, as input grid in the grid calculator).

In this case, you should change the size of one of them so both can get into the same project.

Now, consider this situation. You are studying a large area and you have a very precise information of it. The DEM you have has a cell size of only 10 meters, thus resulting in a huge file which almost collapses your PC. As you are developing a macro scale study, you don't need a high precision, and it would be enough for you to have a DEM with a cell size of 50 meters (which would mean a file 25 times smaller and a great relief for your CPU).

The problems depicted in the above situations can be solved simply resampling the grid, but, since each case is different, the parameters and methods used to perform this resampling are different. Select the *Tools/Resample Grid* to start working.

This first parameters window (and I say *first* because this time there are a couple more yet to come...) is quite simple.

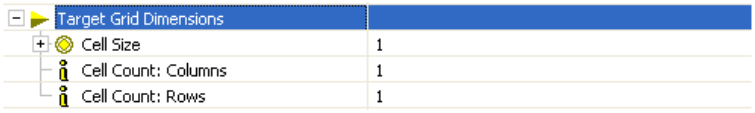
Select the grid you want to resample in the *Input* field and then the characteristics of the new grid to be created (the resampled one). The following options are available:

- **Specify dimensions**: The user can specify the new cell size in the current grid units. The resulting grid will cover the same area but with a different number of cells. This should be used in the second of the above depicted cases.

- **Create new grid in existing project**: Use this to move grids between projects, as needed in the first situation depicted above. The resulting grid will have the same characteristics as the grids of the specified project. That means that not only the cell size will be modified, but also (if needed) the boundaries of the grid will be change to fit into the project.

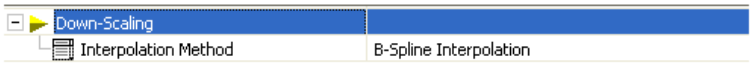
- **Overwrite existing grid**: Same as above, but instead of creating a new grid in the project it will overwrite an already existing one.

For our fist try, we will choose the first option, *Specify dimensions*. Press the *OK* button and you will see the following parameters window:



Only the first field is editable. Enter in this field the cell size you wish for the new grid. Once you have typed it in, press enter or click with the mouse somewhere else, so SAGA can process the introduced value and update the fields containing the number of rows and columns of the new grid. The lower the number you introduce, the greater the number of rows and columns, and viceversa.

Once you have introduced the desired cell size, press the *OK* button to get to the third (and last) parameters window.



Only one field can be found here: *Interpolation method*. If you are down–scaling the grid (that is, resampling the grid to a new one with a smaller cell size) you will have five options to choose from:

- Nearest Neighbor
- Bilinear Interpolation
- Inverse Distance Interpolation
- Bicubic Spline Interpolation
- B–Spline Interpolation

These are the methods that SAGA will use to assign data to all those cells that did not exist in the original grid (since the resampled grid is bigger, it will have more cells)

When you are up–scaling (that is, creating a new grid with a larger cell size), a new option will appear:

- **Mean value**

SAGA will use this method also to calculate the values of the cells in the new grid. Since the grid is now smaller, the space occupied in it by one cell was occupied by more than one cell in the original one. Using the values of this cells, a new value is obtained according to the selected method.

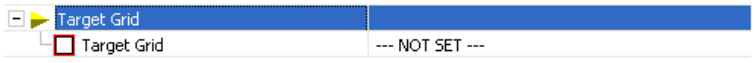
I will not explain here the characteristics of all this methods. Some of them are more time consuming and yield better results, while some are preferred in some circumstances. Again, the differences between them can be better understood using digital images than grids, so if you have a background in digital image processing you will probably know them already.

There is, however, one difference that I would like to point out, because it is really important, since not knowing it can lead to really wrong results. All the methods except for the *Nearest Neighbour* one can introduce values in the resampled grid that are not found in the original one. That means that if you have two cells with values 1 and 2 and a new value has to be calculated for a cell between them, this value can equal 1.5, even if all the cells in the original grid are integer values. This is interesting when dealing with grids that represent continuous variables, but it's not a good thing when you are working with discrete grids.

Imagine that the previous grid is a land–use grid and value 1 represents forested land, while value 2 represents urbanized land. What does value 1.5 means? Nothing. Classes are represented by integer values, and an integer value should go into each cell of the resampled grid. So whenever you work with grids containing classes instead of actual values, always use the *Nearest Neighbor* method to keep the coherency of the new grid.

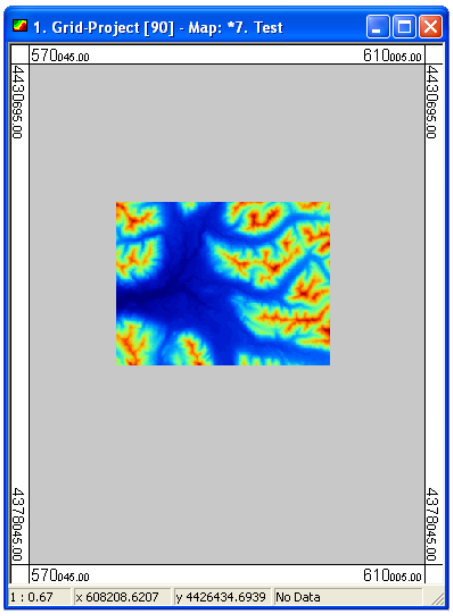
Going back to the first parameters window, let's see this time how to create the resampled grid using the dimensions of an already existent one. But before we keep going, do the following: use the zoom tools to select a region within the test.dgm grid and save this region using the *Save Zoomed Area as New Project...* menu item. A new project will appear containing this portion of the test.dgm grid. We are going to introduce this portion into the project in which the test.dgm grid is contained.

Run the *Grid Resampler* module again and select the portion of test.dgm grid as input and the *Create new grid in existing project* method in the *Target Grid* field. Press the *OK* button and you will get to the following parameters window:



Now simply choose from the list the grid whose dimensions you want to taken for the resampled grid. Not only its dimension are used, but also its coordinates and its extent. Cells in the resampled grid that don't overlap with the original grid will get a no data value. Therefore, if the original grid and the grid you select are not in the same geographical area (they do not overlap at all) you will get just a no data grid.

Press the *OK* button and you will get to the third parameters window, which is the same for all the target grid options. Select the interpolation method you prefer, and close the window using the *OK* button. If you now go to the project where the test.dgm grid is located, you will find a grid like this.

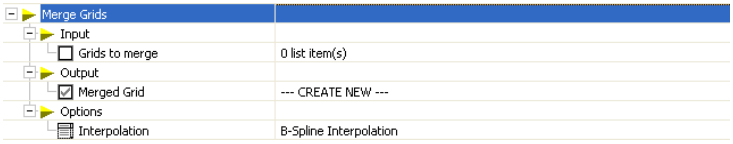


7.10 Merging grids

Quite often, the information about a geographical area is scattered in several files, specially if this area is large. If you have to perform some kind of analysis on this area, you might want to gather all this information into just one file, so it is easier to use it. Sometimes this is even a must, for example in the case of hydrological analysis, since the connection between different grids is important to extract results.

To merge several grids, you first have to load them. They will be located in separate projects, as their extents are different, but they must have the same cell size. Therefore, if they have different cell sizes, you must use the *Grid Resampler* module to homogenize them.

To start working, create a couple of new projects selecting portions of the test.dgm grid and then using the *Save Zoomed Area as New Project...* command. Select the *Merge Grids* menu item.



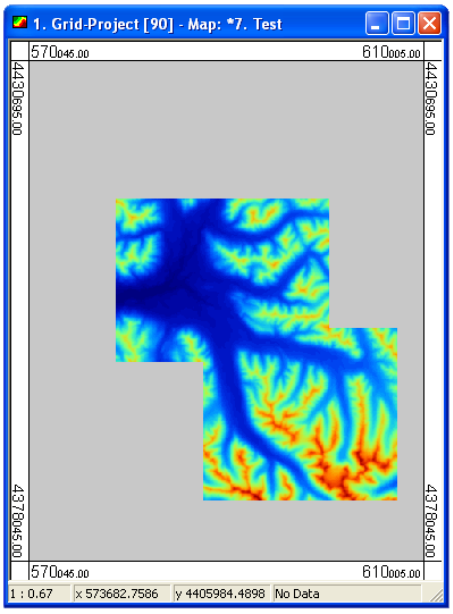
Introduce as input all the grids that you want to merge, and select an output grid. If you select the — *CREATE NEW* — item, the new grid will have the smallest extent needed to include all the input grids. If you select an already existent grid, the new grid will have its same extent and grid size.

Be careful when using the — *CREATE NEW* — option, because if the input grids are far from each other, the minimum grid needed to enclose them might be too large, even causing your computer to freeze.

When using an existing grid as output, the cells size of this grid is used, and some interpolation will be performed if input grids have a different cell size. What was said about this subject in the last chapter applies here as well.

Also, when using this option, the input grids can have different cell sizes.

Merging a couple of portions of the test.dgm grid will yield a result similar to the one below.



See how some cells have been filled with no data values (the gray ones).

7.11 Filling missing information

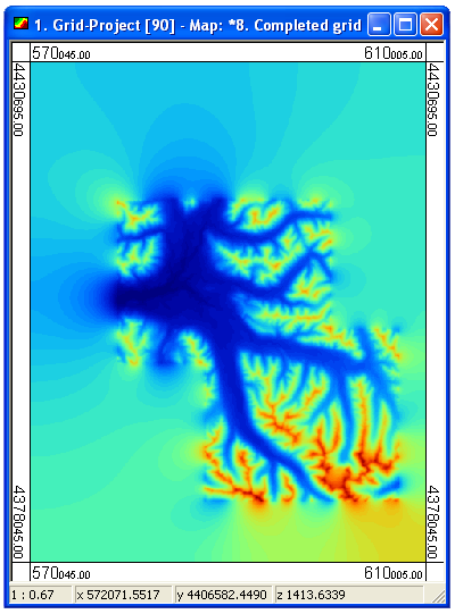
The modules under the *Grid/Tools/Gaps* menu can be used to complete grids that are missing information in some of their cells. The *Close Gaps* and *Close One Cell Gaps* will perform this completion using solely the information contained in the grid to complete, while the remaining one, *Grid Completion*, will use and additional grid.

The parameters window of the *Close One Cell Gaps* module is very simple, just containing the *Input* and *Output* fields. Use this module when isolated cells of a grid contain no data values and you want to replace them with new values obtained from the surrounding cells.

If the areas missing data are larger that just a single cell, you should use the *Close Gaps* module. In its parameters window, along with the *Input* and *Output* fields you will find one named *Tension Threshold*. This parameters controls how the available information is used to fill the cells with no data values. Try different values until you find the one that suits you. Larger values take a shorter processing time.

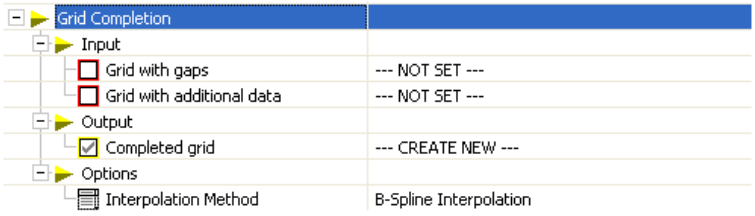
When you use this module, SAGA goes through several processing stages. You can see its progress in the grid window, which gets updated at each stage.

Taking the grid that we obtained as a result from the *Merge Grid* module (which contained large no data areas) and completing it using the *Close Gaps* module, we get something like this:



Clearly, the no data areas are way too large, but with smaller ones the results can be quite satisfying. Also, no data areas surrounded by cells containing valid information generate better results than areas located in the outside, as in the example grid.

To perform a more complex grid completion, use the *Grid Completion* module.



Using this module, the additional information for the no data cell is not taken for the grid itself, but from an additional one. Select the grid with missing data in the *Grid with Gaps* field, and the one containing additional information in the *Grid with additional data* field. Select an output in the *Complete Grid* field and press the *OK* button.

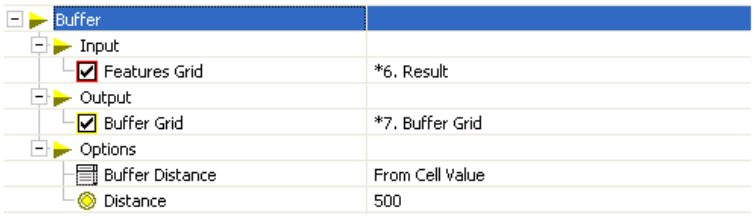
7.12 Creating buffers

Buffers are one of the most useful functionalities found in GIS applications. They are usually conceived as a vector operation, but they can also be created using raster layers.

To create a buffer you will need a raster layer containing feature information. The features are represented with cells containing any valid value except zero and the remaining cells not to be buffered must have a no data value or a zero value. With this information, the buffering module creates a new grid in which features cells are assigned a value of 1, cells that are included within the buffer are assigned a value of two, and the remaining one remains as no data cells.

We haven't used yet any features grid, so the first thing to do is to load one. Go and load the testchan.dgm grid included in the demo.zip file. This is the same as the test.shp file, which contained a channel network, but in raster format.

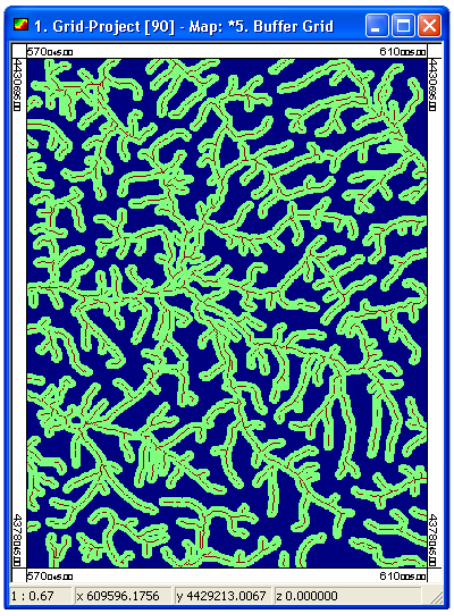
Let's calculate an influence area on both sides of these channels. Select the *Grid/Tools/Buffer* menu item.



The influence area we are about to create has to be defined, introducing the distance to be put around those cells that contain feature information (cells with non–zero values that do not equal the no data value). There are two ways of introducing this distance: using a fixed one or one that depends on the value of each cell.

To use a fixed distance, set the *Buffer Distance* field to *Fixed* and type in the distance in the *Distance* field. It should be expressed in the same units as the cell size (usually meters). Using this method, SAGA includes in the buffer all those cells that are within the specified distance from a feature cell. The buffer analysis is performed in a per–cell basis, even when the features represented are polygons or lines.

Choosing a distance of 500 meters you should get the following grid:



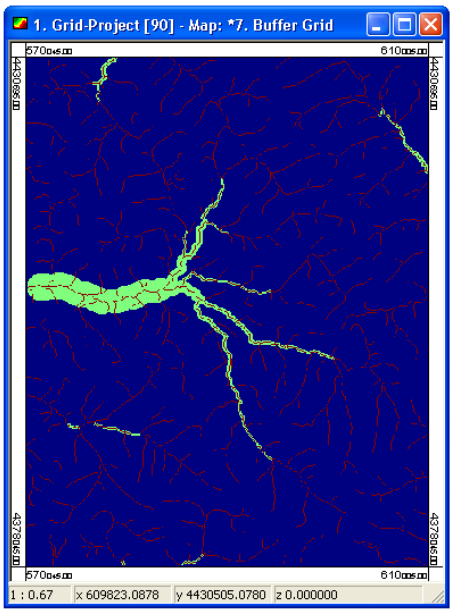
Along with this method, SAGA allows to define a different buffer for each cell, so some of them can have larger influence areas according to their characteristics. In the example grid (testchan.dgm), each cell contains the order of the river that flows through it, higher values thus representing more important rivers. We can use this values to assign larger buffers to those rivers of a higher magnitude.

Select the *Cell Value* item in the *Buffer Distance* field. The value found in each feature cell will now be used to create a buffer around it. In the case of the testchan.dgm grid, however, selecting this option will cause the resulting grid to be practically identical to the original one, and almost no buffers will be created. Why? The values in the testchan.dgm grid, which represent orders, range from 1 to 129. Consequently, buffers created using this values will never be wider than 129 meters. Since cell size in this example grid equals 90 meters, the width of the buffer will be almost always less than one cell and, therefore, no buffer will be created around most cells.

Try using the grid calculator to create a new grid that you can use to define the influence area. For example, multiply the testchan.dgm grid by 10, and you will get a 200 meter wide

buffer (200 meters on each side, 400 meter total width) in channels with an order of 20, 500 meters for those with an order of 50, and so on.

The resulting grid from this will look like the one shown below:



7.13 Creating artificial grids

Sometimes it might be useful to be able to generate artificial grids. The modules *Grid Function* and *Random Terrain* can generate *mathematical* terrains or real looking ones from scratch (i.e without needing any other grid), that can be used for different purposes.

From a practical point of view, you will probably never need to use these modules, but they are quite interesting from a scientific point of view. As such, we will later use the results of this modules to test the algorithms implemented in the terrain analysis library.

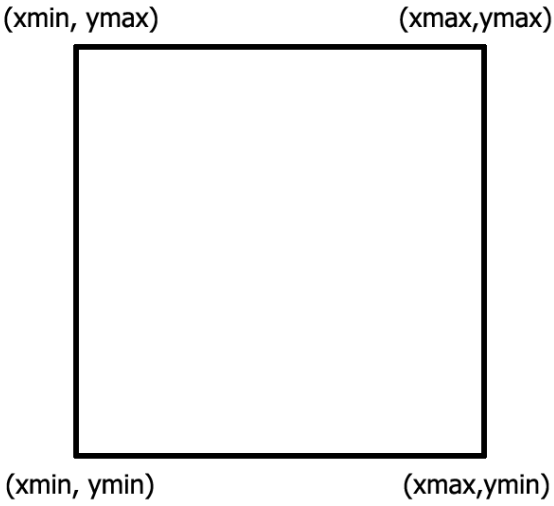
Starting with the first one, select the *Calculus/Grid Function*.

This module will create a grid using a formula that will give each cell a Z value depending on its x and y coordinates. That is,

$$z = f(x, y) \tag{7.6}$$

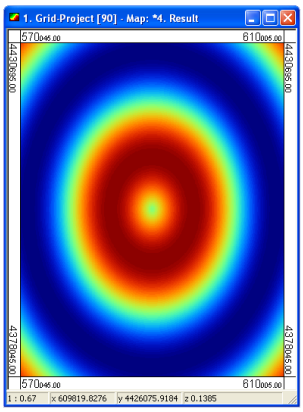
To define this formula you can use all the functions already explained in the grid calculator section, even the boolean ones!! The difference here is just that the Z values are not calculated using the value of the cell in a grid, but its position within it.

Since you already know how to use formulas in SAGA, the only thing you have to learn is how SAGA calculates the x and y coordinates of a cell. Using its "real" UTM coordinates? Counting the number of cells from a reference point? None of them. To illustrate how this calculation is performed, have a look at this very simple graph:

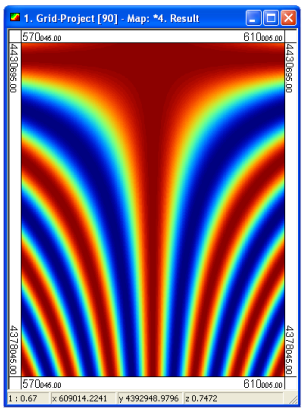


There are four boundary values, namely xmin, xmax, ymin and ymax. Using this values (which correspond to the boundary cells indicated in the corners of the above image) SAGA calculates the remaining ones just interpolating them. By changing this four values you can get different results using the same formula.

Here you have a couple of examples that will help you understand how this module works:



xmin = -5
ymin = -5
xmax = 5
ymax = 5
formula = sin(sqrt(x*x+y*y))

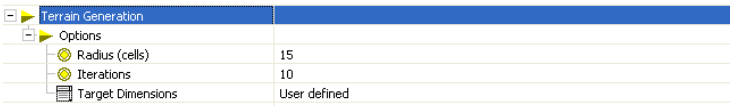


xmin = -1
ymin = -20
xmax = 1
ymax = 1
formula = cos(x*y)

As it was said before, this mathematical surfaces will be used in the terrain analysis chapters, so I will leave you a little homework. Test your maths skills and try to create a semi–spherical suface using the *Calculus/Grid Function* module. Don't worry if you can't, I will give you the formula in a few chapters.

Although this surfaces are quite interesting, they don't look very real, at least if we want to use them as a Digital Elevation Model (fortunately the earth surface doesn't have a regular sinusoidal shape...). If you need a more realistic terrain, the *Random Terrain* module will help you generate it.

Select the *Tools/Random Terrain* menu item. You will see the following parameters window.



Two values control how the terrain is created:

- **Iterations**: The more iterations it makes, the larger time it takes to finish, but the more evolved the resulting terrain is.
- **Radius**: The radius defines the size of the area affected by each iteration. A larger radius implies a larger time needed to complete module execution.

After selecting these parameters, you must define the extent of the grid you are about to create. Three options are available:

- User Defined Grid
- Grid
- Grid Projects

These are very similar to what we saw in the section about grid resampling, so I suppose you will have no problem understanding what each one means.

If you select the *User Defined Grid* option, you will be prompted for the dimensions and the cell size of the new grid.

Since the grid is created from scratch, and no coordinates are introduced in any field, the coordinates of the lower left corner of the grid are set to (0,0). Anyway, coordinates will not probably be needed when working with an artificial grid such as this.

7.14 Some examples

Before we head into the next chapter, let's have a look at a couple of interesting examples. Although you already know how to use modules such as the grid calculator, it's a good idea to study a few real problems and try to solve them using these modules. By doing this, you will learn some tips and tricks about all the functions described in this chapter, that will surely help you solving other problems.

In this section we will learn how to intersect two thematic grids and create a new one from this intersection, and also how to constrain grid data to a region of interest using bitmasks.

7.14.1 Performing grid intersection

For the first case, we will deal with some hydrological concepts that I will briefly introduce before starting. If you are already acquainted with them, skip a couple of paragraphs to get directly into the SAGA stuff.

There are many methods for calculating runoff from rainfall values. A widespread method is the so–called Curve Number method, with evaluates runoff using a value between 0 and 100. 0 indicates that no runoff is generated, while 100 means that the surface is completely impermeable and every single drop of rain will become runoff.

The exact formula to estimate runoff from rainfall value and Curve Number value is:

$$P_E \begin{cases} = \frac{(P-0,2S)^2}{P-0,8S} & \text{if } P \geq 0,2S \\ = 0 & \text{if } P < 0,2S \end{cases} \tag{7.7}$$

where
P represents rainfall,
R runoff,
and S maximum potential retention.

The Curve Number is used to calculate S, according to

$$S = 2,54 \left(\frac{1000}{N} - 10 \right) \tag{7.8}$$

where N is the Curve Number.

Simplyfing a bit, we can assume that the Curve Number value mainly depends on the type of soil and the land–use. There are many available tables that define this dependency, but, to keep this exercise as simple as possible, I will use the following one, which is simply an excerpt from a larger one, adapted to the values in the example grids included in the demo.zip files (these grid files are named CNSoil.dgm and CNLand.dgm)

	Soil Type A	**Soil Type B**	**Soil Type B**
Cultivated land without conservation treatment	72	81	88
Cultivated land with conservation treatment	62	71	78
Woods and forests	36	60	73

The different soil and land use classes are codified in the corresponding grids as follows:

- Cultivated land without conservation treatment = 1
- Cultivated land with conservation treatment = 2
- Woods and forests = 3
- Soil Type A = 1
- Soil Type B = 2
- Soil Type C = 3

Using these two grids (load them in case you have not already done it) and the above table, how can we generate a new grid containing Curve Number values? Clearly, we must use the grid calculator to perform some kind of operation using the soil and land–use grids, but that will not be enough. The question here seems to be: which formula should I introduce in the grid calculator? Well, the key is not in the formula, but in the combination of the formula itself and the values used to code the different soil type and land–use classes.

Imagine that you use the following formula in the *Formula* field of the grid calculator: `a*b` . This is not a good idea, because the combination of a cultivated land without conservation treatment and a soil of type B will yield the same result as the combination of cultivated land with conservation treatment and a soil of type A, that is, 3. That value of 3 is not a Curve Number value, but a class of Curve Number values. If you later want to assign this class a real Curve Number value, which one will you use? 81? 62?

To ensure that a CN class number can only be produced by one (and only one) combination of soil type and land–use, we can change the coding of values and use the following one:

- Cultivated land without conservation treatment = 1
- Cultivated land with conservation treatment = 3
- Woods and forests = 5
- Soil Type A = 7
- Soil Type B = 11
- Soil Type C = 13

By using prime numbers, we avoid coincidences, so there is a biunivocal relation between combinations of land–use and soil type, and the values in the resulting Curve Number class grid.

However, for parameters with a larger number of classes, using prime numbers might not be very handy. Every one knows the first ten or so first prime numbers, but for the next ones you will probably need to use an Erathosthenes sieve or any other similar thing.

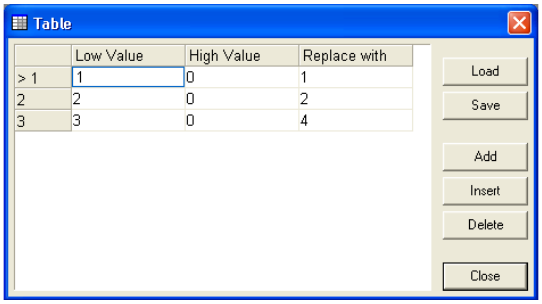
A more practical solution can be found using the formula `a+b` and coding the soil type and land use classes using powers of two (or any other number you want). Have a look at the following tables to see what I mean.

- Cultivated land without conservation treatment = 1
- Cultivated land with conservation treatment = 2
- Woods and forests = 4
- Soil Type A = 8
- Soil Type B = 16
- Soil Type C = 32

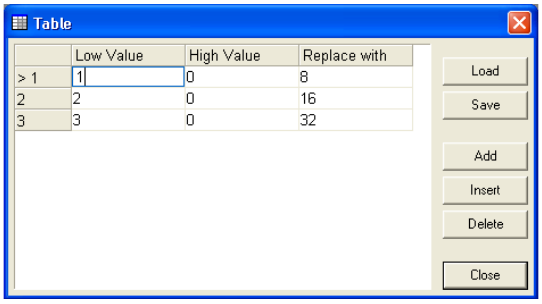
If now you add both grids and you find a value of 20 in a cell, you know that it is a cell with soil type B in which a forest is found (computer geeks will like to write down the number in binary to see more clearly where it comes from), so it should have a Curve Number of 60.

Well, now that you know the theory, let's calculate that CN grid using the land–use and soil type grids form the demo.zip file. We will use the last coding scheme, that is, using powers of two.

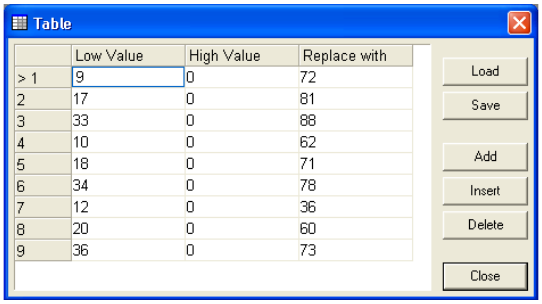
To change the original coding, run the *Replace Grid Values* module and enter the following table in the *Lookup table* field for the land–use grid.



Use the next one for the soil type grid.



Now add the modified grids and you will get a grid containing CN classes. To turn this classes into their corresponding CN values, use the *Replace Grid Values* module again, selecting this last grid as input and using the following lookup table:

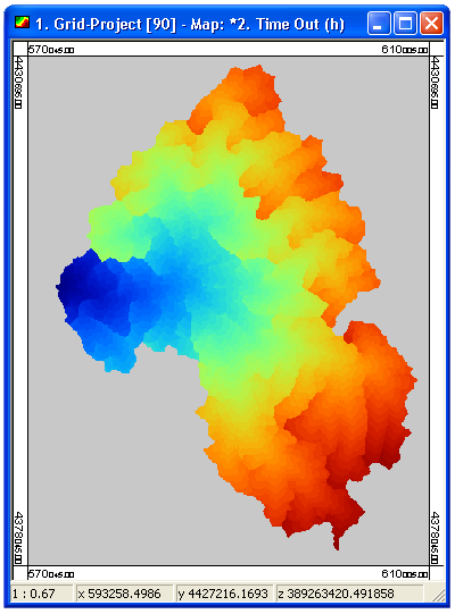


7.14.2 Using masks

Masks can be used to exclude cells from analysis or simply to exclude them from being represented. A very simple way of creating masks is using 1 and 0 values, and also no data values.

Whenever a cell with a no data value is used to evaluate an expression using the grid calculator, this expression always returns another no data value (more precisely, the no data value of the resulting grid, so the cell in that grid is set to no data).

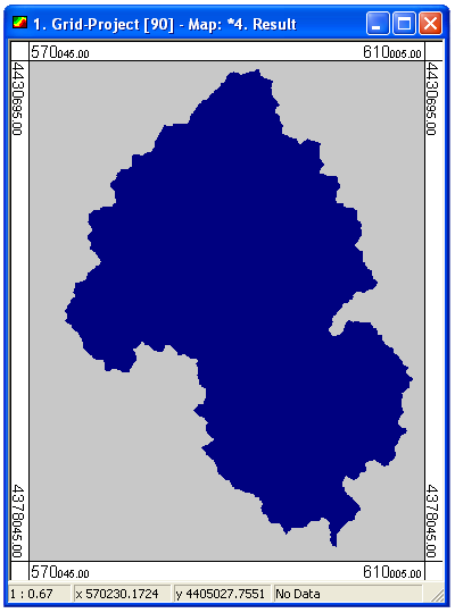
Open the testtime.dgm file included in the demo.zip one. It contains a grid of the same characteristics as the test.dgm grid (and will be, consequently, put into its same project), representing travel time to the outlet of a basin (we will see how to create this grid in a few chapters). All the cells outside the basin are set to no data.



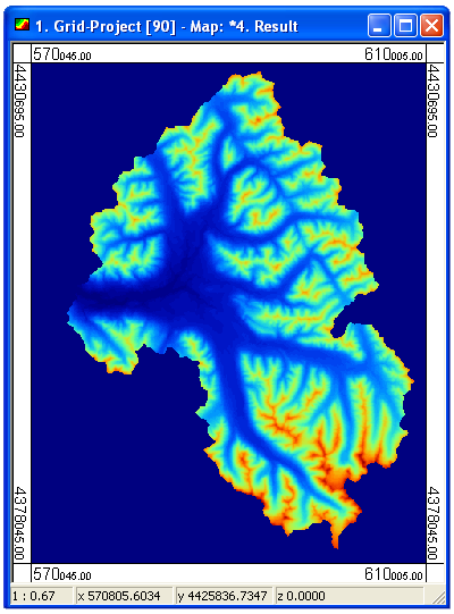
We can use this grid to create a mask and later apply it to any grid, so only those cells located into the basin are shown and the rest of them are ignores. That will result in a nicer appearance (if we are working with a basin, there is no point on showing the outside cells), but also it can cause some modules to execute faster on this grid, since the information in all the no data cells can be neglected when performing calculations.

We will use a mask that, when multiplied by another grid, leaves the cells inside the basin unchanged and turns the remaining ones into no data cells. To get this, we need all the values into the basin to equal 1.

Dividing the time out grid by itself will produce the desired mask. All of the cells outside the basin will become no data cells, since no data values are involved in the calculation. Also, if all the cells outside the basin equaled zero, they would get a no data value in the output grid, since $\frac{0}{0}$ results in no data.



Multiply the test.dgm grid and the mask grid and you will get the following result.



If you change the no data value color to white, the appearance is even nicer.

Chapter 8

Import/Export modules

8.1 Introduction

SAGA IO capabilities have been enough until now, mainly because all the example layers (whether raster or vector) were stored in natively supported file formats. However, specially in the raster case, you are likely to found information stored in many other different formats within your daily work with SAGA. Most of this formats are also supported, but additional modules are needed.

Not only input capabilities are implemented in these modules, but also some output ones. You will find them very helpful, since you might need to use some of the generated grids or vector layers in other applications that do not support SAGA native formats.

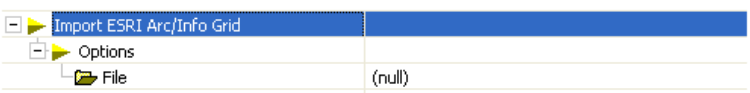
Nevertheless, if you have no need of exporting data, I recommend to use the Shapefile format for vector layers and the DiGeM format for raster ones (they are faster and easier), and use Import/Export modules only when no other suitable alternative can be found.

The number of file formats supported by SAGA via Import/Export Modules is quite large and constantly growing (and, probably, support for more new formats will be added in each new version). Each file format has its corresponding module and all of them behave rather similar, so this chapter will just introduce the main structure of Import/Export modules and add some information about each supported file format, instead of describing each one of them individually.

8.2 Import modules

Most of the import modules deal with raster data, while only two of them can handle vector data. This is quite logical, considering that the most popular vector file format — ESRI Shapefile — is supported by SAGA without needing any module (on the other hand, SAGA native raster format, the DiGeM format, is used exclusively within SAGA, and I do not know of *any* other application that supports this format).

Raster import modules can be found under the *File/Grid/Import* menu. Select the *Import ESRI ArcInfo Grid* menu item to get to the following parameters window.



You will find a *File* field in all the parameters windows of all the import modules, and most of the times nothing else.

Simply click on the *File* field to see a file selection dialog, select the file you want to import, close the dialog and press the *OK* button. The file you selected will be loaded and a new grid thumbnail will appear in the project window.

As you can see, just one file can be opened each time you execute the module. This can be rather awkward if you want to build a big project whit a large number of raster layers, all of them stored in a file format that requires using an Import/Export Module. For that reason, I recommend to save the layer in DiGeM format as soon as you open it, so you can forget about the original file and work just with a new one which SAGA can open "by itself".

Also, have in mind that projects cannot be created with layers that are not saved in SAGA native file formats.

Although they are all very simple, some modules need some additional information apart from the filename. Here is a list of those modules and a brief explanation of what else you have to introduce in their parameters window.

- **Import Surfer Grid**: Information about no data values must be provided.
- **Import Binary Raw Data**: The most complex one. You must know several grid properties such as number of rows and columns, cell size, etc. More likely to use with images. Not for the beginner. If you are using it, surely you know how to do it.
- **Import Windows Bitmap**: Easy...
- **Import USGS SRTM Grid**: Use it to import United Stated Geographical Service files. The resolution of the grid must be defined.
- **Import MOLA Grid**: Wanting to do some terrain analysis in Mars? Use this module to open NASA files with martian elevation models.
- **Import SRTM30 DEM** Use it to import elevation data from NASA's Shuttle Radar Topography Mission. Notice how the *File* field has been replaced with a *Path* field. Select here the path where the uncompressed files can be found. Use the four remaining field to define the extension of the area you want to import.

  SRTM data can be downloaded from:

  ftp://edcsgs9.cr.usgs.gov/pub/data/srtm/SRTM30

- **Import GDAL**: The GDAL library is an open source library that support a large number of image and grid file formats. It has been developed as an independent project, but, since it is open source, it can be incorporated into other programs, such as SAGA. With it you can import the following file formats (yes, some of them can also be imported using other modules).

  - VRT: Virtual Raster
  - GTiff: GeoTIFF
  - NITF: National Imagery Transmission Format
  - HFA: Erdas Imagine Images (.img)
  - SAR_CEOS: CEOS SAR Image
  - CEOS: CEOS Image
  - ELAS: ELAS

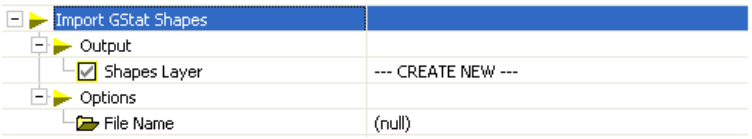
  - AIG: Arc/Info Binary Grid
  - AAIGrid: Arc/Info ASCII Grid
  - SDTS: SDTS Raster
  - DTED: DTED Elevation Raster
  - PNG: Portable Network Graphics
  - JPEG: JPEG JFIF
  - MEM: In Memory Raster
  - JDEM: Japanese DEM (.mem)
  - GIF: Graphics Interchange Format (.gif)
  - ESAT: Envisat Image Format
  - BSB: Maptech BSB Nautical Charts
  - XPM: X11 PixMap Format
  - BMP: MS Windows Device Independent Bitmap
  - PCIDSK: PCIDSK Database File
  - PNM: Portable Pixmap Format (netpbm)
  - DOQ1: USGS DOQ (Old Style)
  - DOQ2: USGS DOQ (New Style)
  - ENVI: ENVI .hdr Labelled
  - EHdr: ESRI .hdr Labelled
  - PAux: PCI .aux Labelled
  - MFF: Atlantis MFF Raster
  - MFF2: Atlantis MFF2 (HKV) Raster
  - FujiBAS: Fuji BAS Scanner Image
  - GSC: GSC Geogrid
  - FAST: EOSAT FAST Format
  - BT: VTP .bt (Binary Terrain) 1.3 Format
  - L1B: NOAA Polar Orbiter Level 1b Data Set
  - FIT: FIT Image
  - USGSDEM: USGS Optional ASCII DEM
  - GXF: GeoSoft Grid Exchange Format

- **Import Image**: This module support several image file formats, including bitmaps and some others also supported by the GDAL library. However, if features an additional functionality: if you set the *Split channels* to true, instead of creating a new grid with the chosen image, it creates three independent grids (all of them in the same project), containing the red, blue and green channels of the image.

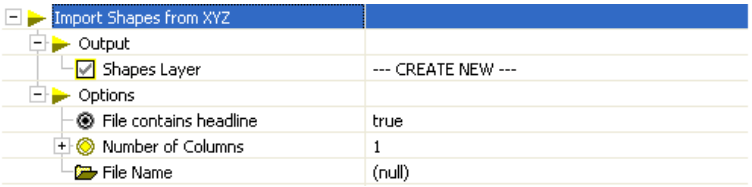
Unlike raster ones, vector import capabilities are rather reduced, but interesting anyway.

When importing a vector layer you will not be prompted just for the name of the file to import. Since several vector entities can coexist within just one vector layer, you will be able to choose whether to create a new layer from the information contained in the imported file, or add it to an already existent one. Also, some additional data might be required depending on the chosen file format, as it happened in the case of raster layers.

Two vector file formats are supported: GStat Shapes and XYZ. The following picture shows the parameters window for the first one of them.



As you can see, it is as simple as possible, so no further explanation is required. If you choose to open a XYZ file, you will find a parameters window like this:



A typical XYZ file is a text file with an structure like the one below.

```
X Y Z
-0.743663 0.401532 0.000000
0.200990 0.512085 0.000000
0.517549 0.255804 0.000000
0.437153 -0.030629 0.000000
0.180891 -0.347212 0.000000
```

It is divided into columns containing X, Y and Z values (some other columns might also be present, but they are optional and depend on the data associated to the layer), and each row contains information about a single point (remember that all vector entities could be reduced to points...).

The first row containing the names of the columns is not always found. If there is a header row, you must tell SAGA to ignore it, setting the *File contains headline* to true.

Introduce the number of columns (3 in the above example) in the *Number of Columns* field, and then enter the number of the columns that contain the X and Y coordinates of each point. This information will be used to locate the points in the view window, and will also appear in the attributes table of the vector layer. Columns are numbered from left to right starting from 1.

Vector layers created with this module are always point layers. If those points represent line nodes, then you will have to run another module to reconstruct the lines, but the import module itself is not capable of creating any other type of vector layer.

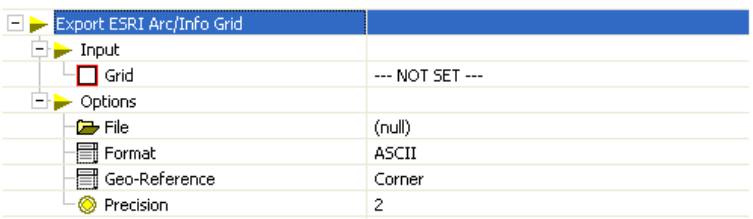
8.3 Export modules

Less numerous than import modules, export modules are pretty similar to those, except maybe for requiring some additional information (not much) to be provided.

Again, we will start with raster ones.

These are the available modules:

- **Export ESRI Arc/Info Grid**:

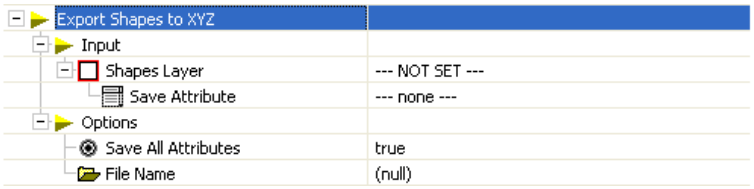


The parameters window is very similar to the corresponding import module, but three new parameters can be found. The first one is the type of file to create, whether binary of ASCII, and the second one the number of decimal places to use when saving ASCII files. ASCII files are plain text files that can be edited using a simple text editor such as Windows notepad and read directly on screen, unlike binary files. A cell must be used to georeference the grid. Either a corner or a center cell can be used for this purpose. To select which one of them to use, select it from the list in the *Geo–reference* field. Pay attention to Z–factors when exporting a grid. Exported grids contain the values of the grid, not the ones you see after applying the Z–factor. Therefore, if you export a slope grid and later import it, since the default Z–factor is 1 you will get the imported grid in radians. Be sure to change the Z–factor if you want to have your grid again in degrees.

- **Export Surfer Grid**: Just like the import module, but you can choose whether to create a binary or an ASCII file.
- **Export Grid to XYZ**: Export the grid as a collection of points with Z values, one for each cell. Caution: very large files might result from using this module. Set to true the *Write Field Names* to include a header describing each field.
- **Export Image** Using this module you can save the grid as an image, in other formats rather than the awkward Windows bitmap format. The whole grid is saved, there's no option here to save just the zoomed area.

As for vector export modules, they are also very simple and do not need much explanation. These are the available vector export modules:

- **Export GStat Shapes**. Exactly like the import one.
- **Export Shapes to XYZ**:



If you set the *Save All Attributes* filed to true, all fields in the attributes table of the exported layer will be saved. Otherwise, only the field selected in *Save Attribute* will be added to the X and Y values of each point.

- **Export Shapes to Generate**: This file format is not supported by any import module. Only a field of the attributes table can be exported. Select it from the *Field* list.

Chapter 9

Terrain analysis. Hydrology. Lighting

9.1 Introduction

It is not venturous to say that we are now in the most important chapter of the book or, at least, the most genuine one. Terrain analysis is a key tool in GIS applications, but is even more important in SAGA, since SAGA comes from a terrain analysis tool such as DiGeM and it is itself a very powerful software to work with Digital Elevation Models.

Terrain analysis tools are included in the Terrain_Analysis.mlb module library. However, not only these modules are described here. Since some of them deal with hydrological concepts, other modules for hydrological analysis have been also included in this chapter, so as to keep them together. You will find them in my Hydrology.mlb library, which can also be downloaded from the SAGA website.

As a consequence of all this, be ready for a rather dense and long chapter, but also be prepared for some of the (IMHO) most interesting stuff presented here.

Terrain analysis modules not only feature some very important functionalities, but are also built using module capabilities that have not been described yet. Therefore, this chapter will also introduce you some new characteristics and elements of modules that go beyond the simple parameters windows that you already know.

Having an overall understanding of the algorithms implemented in these modules and knowing the meaning of the resulting grids is here more important than it was in other chapters, so more detail has been put in describing the elements that will appear through the following section. Read this descriptions carefully, since it is quite easy to misunderstand some of the terrain analysis features and get wrong results from them.

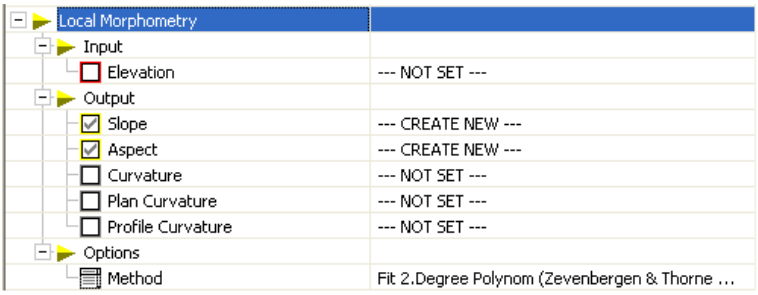
The only thing you need to work with this modules is a DEM. You already have one, the test.dgm grid. You can unload the remaining example files, since we are not going to use them in this chapter.

And now, let's get down to the nitty–gritty.

9.2 Basic morphometric analysis

We will start from the ground up to the most complex modules and functions. That does not mean, however, that this first modules are less useful that the ones yet to come. In fact, the first new grids we are going to calculate (slope and aspect) are among the most important and most frequently used of all the ones that we are about to create in this chapter.

Under the *Terrain Analysis/Morphometry* module, you can find several menu items. Among them, *Local Morphometry* is by far the one you are most likely to use in a regular terrain analysis session. Select it and you will get to the following parameters window.



As it has been said, no more than a DEM is needed, which must be supplied to the module using the *Elevation* field. Several grids can be created as output, two of them already set to — *CREATE NEW* — by default. Since we want to study the whole module and all its possible results, set the remaining output fields to — *CREATE NEW* — too.

The last field (and the most important one) can be modified to select the desired method that SAGA will use to perform the morphometric analysis. The following methods are available.

- Maximum Slope (Travis et al. 1975)
- Maximum Triangle Slope (Tarboton 1997)
- Least Squares Fit Plane (Costa-Cabral & Burgess 1996)
- Fit 2.Degree Polynom (Bauer, Rohdenburg, Bork 1985)
- Fit 2.Degree Polynom (Heerdegen & Beran 1982)
- Fit 2.Degree Polynom (Zevenbergen & Thorne 1987)
- Fit 3.Degree Polynom (Haralick 1983)

The one set by default (Fit 2.Degree Polynom (Zevenbergen & Thorne 1987)) is usually a good choice. In practice, there's not a great conceptual difference between the last four ones, so it's up to you to choose any of them.

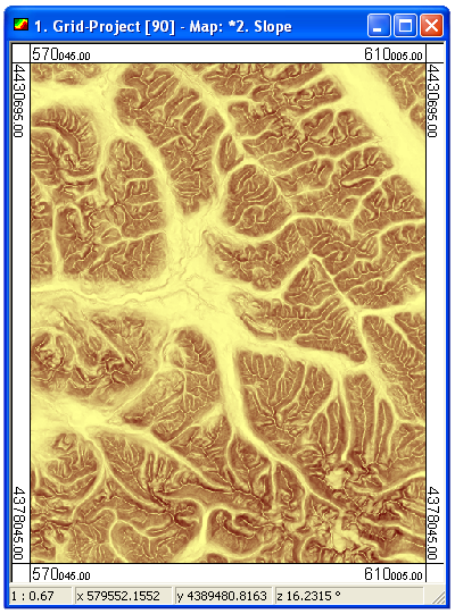
The first two methods are, however, a bit different. They are associated with flow routing algorithms and not just with morphometrical analysis. Due to this, they do not describe local morphometry using a bidimensional function $z = f(x, y)$, and are, therefore, less suitable for some operations. Slope and aspect can be used for all purposes, but curvatures might not be calculated with accuracy, since those methods were not originally conceived having them in mind.

Use this two first methods when you are using its corresponding flow routing algorithms, but if you are just interested in local morphometry, better use one of the others.

Once all the parameters are set, press the *OK* button and you will have five new grids in the grid project window.

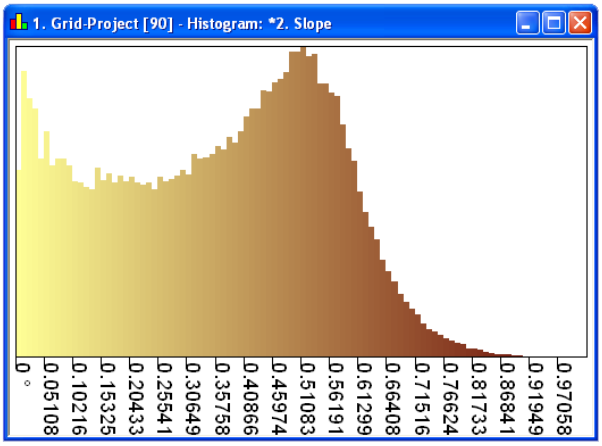
We will carefully analyze each one of them.

The first one is the slope grid.



If you open the settings window of this grid, you will see that its values range from 0 to 58.537.

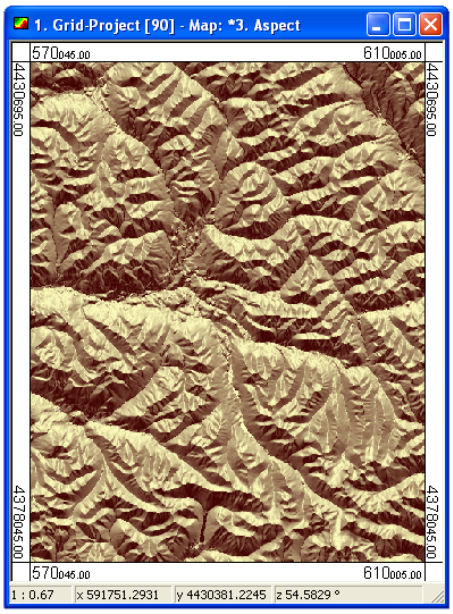
Now have a look at the histogram of the above grid.



Can you see something strange? The values in the histogram are all lower than 1!! Don't worry, there is an explanation. Values of the slope grid have been calculated in radians. However, radians are not very useful, since it is more intuitive to have them expressed in degrees or in percentage. To change the units from radians to degrees, the Z factor is used, and a value of 57.2958 (180/PI) has been introduced in the corresponding *Z factor* field (go to the settings window to see it).

The values in the grid are in radians, but you see them in degrees in the view grid window.

The same Z factor is used in the aspect grid.

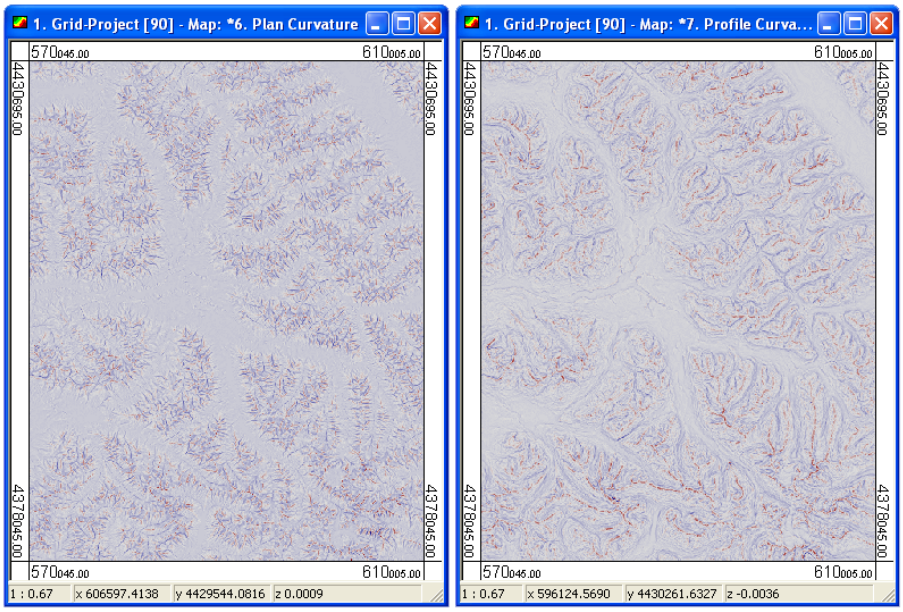


Values in this grid indicate orientation, measured clockwise from North. The grid view looks quite like a shaded grid, but it is not. Hillshade grids and aspect grids share some things in common but are not the same thing. You should have a look at the color palette used with the aspect grid to understand why it looks like that.

Slope and aspect don't require further explanation. From a mathematical point of view, they are calculated using the first derivate, and their meaning is rather intuitive.

The remaining grids that we have created are calculated using the second derivate, and their meaning is not so straightforward. They involve more complex mathematical ideas, and can be interpreted in different ways.

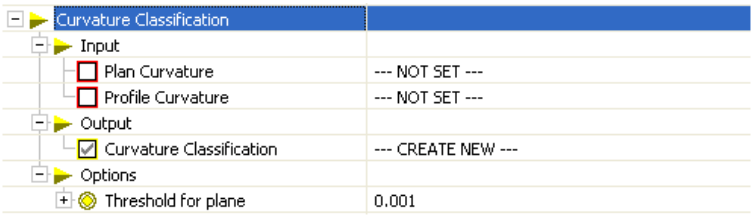
The second derivate of a function can be used to know if it is concave or convex at a concrete point. When working with a surface it can be calculated in any direction, yielding different results. The two most important directions are the maximum slope direction and the one perpendicular to this. The values obtained from this are, respectively, the profile curvature and the plan curvature.



Positive values describe convex curvature, negative values concave curvature.

How to interpret this values? Concavity and convexity can be associated with flow accumulation and dispersion, respectively, so combining the values of both curvature grids you can get a basic idea of how flows behave. This values can also be used to extract some conclusions regarding erosion patterns and other similar physical processes but, since there are other more precise methods to do this which will also be described in this chapter, it is not a good idea to explain this subject with greater detail.

Using the quantitative description of curvature provided by the plan and profile curvature grids, a qualitative description can be obtained using the *Curvature Classification* module.

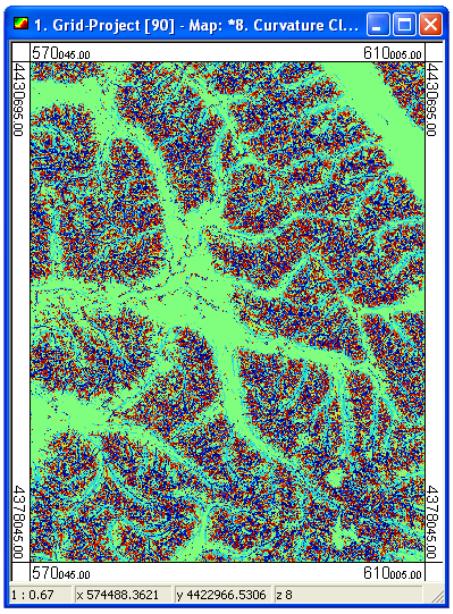


Introduce the plan and profile curvature grids as input and specify a threshold for plane cells. Curvature values under this threshold are considered plane (in the direction of the curvature).

With these values, SAGA generates a new grid in which cells are divided in 9 different classes. The meaning of each class is described next.

- **0**: Plan curvature: concave. Profile curvature: concave.
- **1**: Plan curvature: concave. Profile curvature: plane.
- **2**: Plan curvature: concave. Profile curvature: convex.
- **3**: Plan curvature: plane. Profile curvature: concave.
- **4**: Plan curvature: plane. Profile curvature: plane.
- **5**: Plan curvature: plane. Profile curvature: convex.
- **6**: Plan curvature: convex. Profile curvature: concave.
- **7**: Plan curvature: convex. Profile curvature: plane.
- **8**: Plan curvature: convex. Profile curvature: convex.

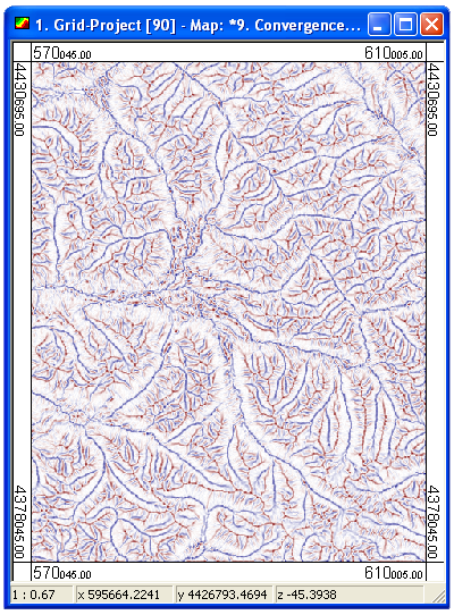
The resulting grid is shown below.



Another similar grid that can be calculated using the morphometry modules is the convergence index grid. The convergence index gives a measure of how flow in a cell diverges (convergence index < 0) or converges(convergence index > 0). These convergence/divergence behavior of flow can be studied using curvature values, as we have already seen, but the convergence index provides a rather more handy way of doing it.

To use this module, you simply have to provide a DEM as input and select one of the two available methods: aspect and gradient. Differences between these methods are not really significant (here is a little exercise you can try: evaluate the difference between aspect and gradient methods using the modules you already know).

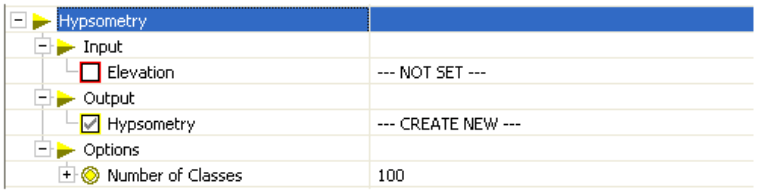
A convergence index grid looks like this.



9.3 Creating an hypsometric curve

Until now, we have only seen modules that created grids as output. The hypsometry of a grid, however, is not something that can be expressed using a grid, but needs to be given in some tabular or graphical form. Consequently, the *Hypsometry* module creates a table instead of a grid, as we will now see.

Run this module selecting the *Morphometry/Hypsometry* menu item.



Select the test.dgm grid in the *Input* field and set the *Output* field to — *CREATE NEW* —. The *Number of Classes* field defines the number of rows that the resulting table will have. The higher value you enter, the more accurate the hypsometric curve will be.

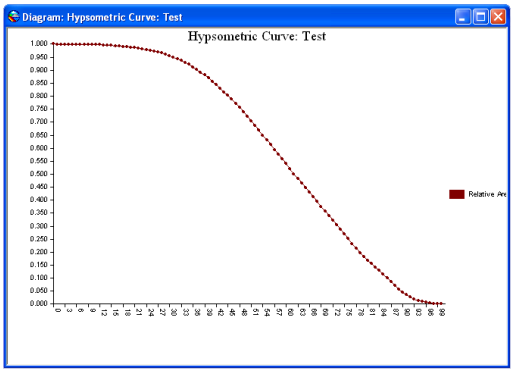
After setting this parameters, press the *OK* button. Apparently, nothing has happened, but if you click now on the *Table* tab in the projects window, you will see a table named *Hypsometric Curve*. Open it.

Table: Hypsometric Curve: Test

	Relative Height	Relative Area	Curve Slope
> 1	0.000000	0.999988	0.000001
2	0.010000	0.999965	0.000002
3	0.020000	0.999939	0.000002
4	0.030000	0.999904	0.000002
5	0.040000	0.999827	0.000005
6	0.050000	0.999747	0.000005
7	0.060000	0.999624	0.000008
8	0.070000	0.999463	0.000011
9	0.080000	0.999187	0.000018
10	0.090000	0.998815	0.000025
11	0.100000	0.998432	0.000025
12	0.110000	0.997929	0.000033
13	0.120000	0.997446	0.000032
14	0.130000	0.996698	0.000050
15	0.140000	0.995835	0.000057
16	0.150000	0.994731	0.000073
17	0.160000	0.993485	0.000083

Here is the information about how heights are distributed in the grid. You can also create it (with much more work, of course, and using a spreadsheet) from the frequency histogram.

To see the Hypsometric Curve (and not just its associated table), create a diagram using only the *Relative Area* field. You should get something like this.



I recall here the usefulness of grid masks. Having an hypsometric curve of a grid is not very useful, but the hypsometric curve of a basin is one of the most classical elements used to describe it. As we saw in the last chapters, values outside the basin can be neglected using a bitmask, thus turning a rather useless result into a really interesting one.

9.4 Classificating terrain forms

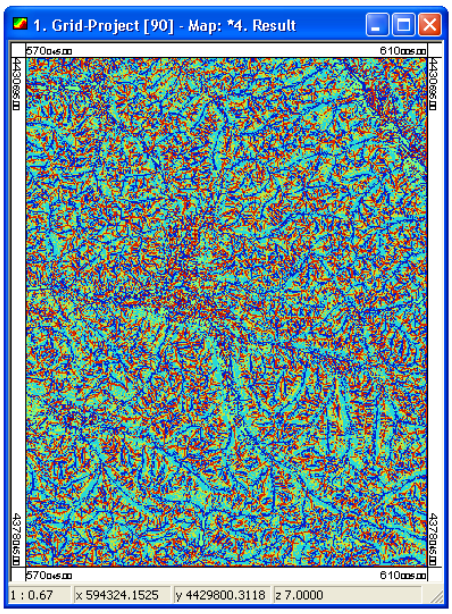
If you take a cell from a DEM and its 8 neighbor cells, you can classify the morphology the center cell according to the elevation values of all of them. Comparing those values, a cell can be assigned one of the following 7 categories of morphometric features (try to figure out why those value...):

- 9. Peak
- -9. Pit
- 1. Pass
- 2. Convex break
- -2. Concave break
- 7. Ridge
- -7. Channel

Creating a grid with those values can be accomplished using the *Indices/Surface Specific Points* module.

Simply select a DEM as input and then set the *Method* field to *Peucker & Douglas*. In the *Threshold* field introduce the threshold that will divide convex breaks from concave ones.

Press the *OK* button and you will get a grid like the following one.



As you have noticed, there are other available methods in this module, but the *Peucker & Douglas* method is probably the best known DEM—based one for identifying morphometric features.

If you try other methods you will get different classifications, all of them being simpler and, at the same time, not so easy to interpret or to use for practical purposes.

As an example, here is the meaning of the values of the *Mark Highest Neighbour* method (directly taken from the module source code):

- 2. Saddle
- 1. Bottom line
- -1. Divide
- Nothing

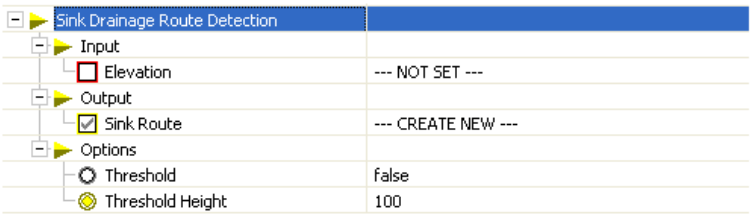
9.5 Preparing a grid for hydrological analysis

From this point, we will start analyzing the hydrological modules that are found in the Terrain_Analysis.mlb and Hydrology libraries. However, DEMs are not always prepared for hydrological analysis, so before we can start extracting hydrological information from a DEM, some modifications have to be usually done to it so as to ensure that other modules (which we will shortly see) will not yield wrong results.

The main source of these wrong results are pits, that is, cells or groups of cells that are surrounded by others with higher elevation. Since there are no lower cells through which route the flow, flow routing algorithms behave badly in this cells. They have to be filled before using any module that involves flow routing, a task that can be accomplished using the modules located under the *Terrain Analysis/Pre–processing* menu.

There are two main ways of preprocessing a DEM. You can create a new standalone DEM with all those pits and sinks "corrected", or create a supporting grid with the flow directions of the cells that comprise the sinks and later use both as input for the flow routing algorithms.

To create the last one, select the *Sink Drainage Route Detection* menu item. You will see the following parameters window.



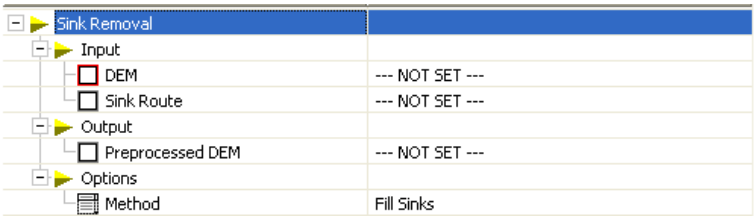
Select the DEM in the *Elevation* field and another grid in the *Sink Routes* field. Do not select the same DEM, since both the original DEM grid and the sink routes grid have to be used together.

A height threshold can be introduced in the *Threshold Height* field (select the *Threshold* field to to true to use it), but you are not likely to use it, since it is not specially useful. Better leave it unchanged.

Press the *OK* button to start module execution.

The resulting sink routes grid will be almost completely filled with 0 values, corresponding to areas without sinks that will not cause any problem to flow routing algorithm. The remaining ones, those cells located within sinks, are assigned an integer value ranging from 1 to 8, which indicates the flow direction that has to be taken from that cell to abandon the sink. This information is used by the flow routing algorithms whenever they cannot calculate one just using elevation data from the DEM.

To create a standalone preprocessed DEM that can be used for hydrological analysis without needing a grid like the previous one, select the *Fill Sinks* menu item.



Simply introduce the DEM you want to preprocess in the *Elevation* field and select the output one in the *Preprocessed DEM* field. To avoid confusions, it is a good idea to select the original DEM as output, so it gets overwritten and you can only work with the corrected one. In fact, in this case, if you set the *Preprocessed* field set to *— NOT SET —*, the preprocessed grid will be written into the original DEM grid.

The most usual way to correct sinks is filling them, but you can also choose to create drainage routes through them, "excavating" the borders of each sink. To select between one of these options, use the *Method* field. For practical purposes there is not a significant difference between them in most cases.

Optionally, a grid with sink routes such as the one created before can be used, selecting it in the *Sink Routes*. It will help SAGA to decide how to fill or "excavate" the sinks.

9.6 Calculating catchment area

Now that we have preprocessed our DEM and it can be used by flow routing algorithms, it is time to get into some serious hydrological stuff. SAGA's powerful raster analysis engine is at its best when dealing with flows and other hydrological concepts, surpassing every other GIS in its capabilities and its usability. However, before using the modules in which the aforementioned algorithms are implemented, it is a good idea to introduce some basic ideas about them.

Flow routing algorithms constitute the key element of hydrological analysis, and this is, consequently, a largely developed area. Many different alternatives exist, each one of them with its pros and cons. Explaining them in detail will take too long, and I will just present those ideas that are more interesting from a practical point of view, so you can have a notion of the difference it makes using one or another method.

Basically, methods can be divided in two groups: Those who consider the flow to move between cell centers and those in which the flow moves "freely" around the DEM (known as Flow Tracing algorithms). The ones in the former group are related with the D8 method (the oldest and the only one that you will find in other GISs) while the ones in the latter are quite more complex and its use is rather restricted.

Another division can be made separating those who consider an unidimensional flow (commonly referred as Single Flow Direction algorithms) and those who consider a bidimensional one (Multiple Flow Direction Algorithms). Attending to these groupings, we can describe the modules implemented in SAGA, namely

- **Deterministic 8 (D8)**: The classical. Flow goes from the center of a cell to the center of one(and only one) of the surrounding cells. Flow directions are, therefore, restricted to multiples of 45º, which is the main reason for most of the drawbacks of the method. (O'Callaghan & Mark 1984).

- **Rho8**: Same as above but with an stochastic component that should improve it. The flow direction is determined by a random argument that is dependent on the difference between aspect and the direction of the two adjacent neighbor cells. Not very useful... (Fairfield & Leymarie 1991).

- **Deterministic infinity (D∞)**: Flow goes from one cell to two contiguous surrounding cells, thus considering a bidimensional flow and overcoming the drawbacks of the D8 method. (Tarboton 1998).

- **Braunschweiger Digitales Reliefmodell**: Another Multiple Flow Direction algorithm. The flow is splitted between the surrounding cell whose orientation is nearest to the aspect of the center cell and its two adjacent cells. (Bauer, Bork & Rohdenburg 1985).

- **FD8** (found in SAGA simply as **Multiple Flow Direction**): A D8–derived bidimensional flow routing algorithm. (Quinn *et al* 1991).

- **Kinematic Routing Algorithm (KRA)**. An unidimensional flow tracing algorithm. Flow behaves like a ball rolling down the DEM, without restricting its position to the center of cells. (Lea 1992).

- **Digital Elevation Model Network (DEMON)**: The most complex one. A bidimensional flow tracing algorithm. Rather time consuming. (Costa-Cabral & Burgess 1994).

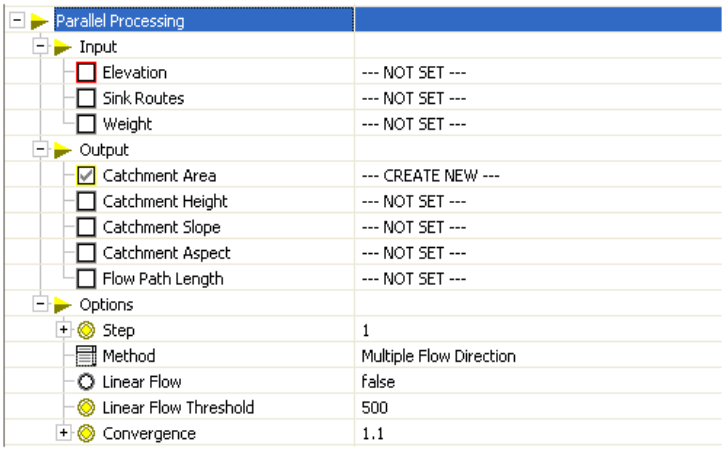
With these information it should suffice to start working with hydrological modules. As we use each method, you will understand the differences between all of them and find which one is better for you to use. For further information, try the references shown above.

Flow directions for each cell are not very useful itself (also, some methods don't use a flow direction in a strict sense), but can be used as a basis for more interesting result. The most important parameter that can be derived is the catchment area, also know as flow accumulation (hence the menu name, as we will see). The catchment area of a cell indicates the area upslope that cell whose flow will eventually reach it. As we will shortly see, a catchment area grid is needed as input in a large number of the modules described in this chapter.

The methods used to generate a catchment area grid are different depending on which flow routing algorithm is used. Therefore, they are implemented in different modules. Three modules can be used to create a catchment area grid, all of them found under the *Terrain Analysis/Flow Accumulation* menu: *Parallel Processing, Recursive Upward Processing* and*Flow Tracing.*

As expected, flow tracing algorithms are found in the *Flow tracing* modules, while the remaining ones (the D8–derived algorithms) are implemented in the modules *Parallel Processing* and *Recursive Upward Processing.* There is no difference in the results you can obtain from the two last ones (except for the fact that one more grid can be calculated using the first one) so I will explain just one of them.

Select the *Parallel Processing* menu item to get to the following parameters window.



Three fields can be found under the *Input* node, only one of them (the DEM) being compulsory. If you have preprocessed the DEM using the *Sink Removal* module, there is no need to use the *Sink Route* field. However, if the DEM is not preprocessed, you should select a grid containing sink routes to avoid errors when running the flow routing algorithms

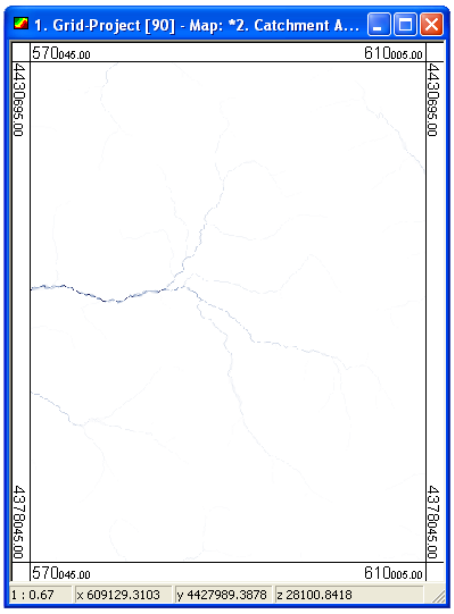
Using the last input field, *Weight*, you can adjust the weight that each cell has when calculating catchment areas. Let's see an example of how this can be used.

Catchment areas are expressed in area units (that is, if cell resolution is expressed in meters, catchment area will be expressed in square meters). If you know the amount of runoff produced in each cell expressed in height units (usually millimeters), you can turn this catchment areas for each cell into runoff volumes that drain to them. However, not all cells generate the same runoff, so each one will have a different weight in the runoff grid. If you have a runoff grid, put it in the *Weight* grid and it will be used to calculate catchment areas.

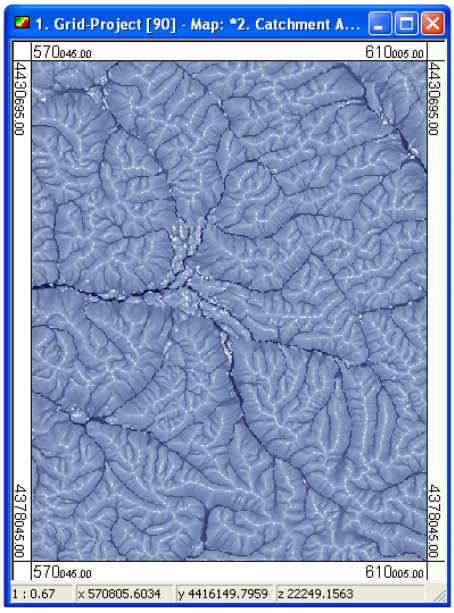
When the *Weight* field is left as — *NOT SET* —, each cell contributes to the catchment area with its own area, which equals the square of its cell size. If a weight grid is used, each cell contributes with the value assigned to this cell in the weight grid. So, if 1000 cells with a value of 2 in that weight grid flow to a cell, this last one will have a value of 2000 in the catchment area grid.

Notice that, actually, the resulting grid is not a a catchment area grid but, in fact, a runoff volume grid, thus having different units (cubic meter instead of square meters, assuming that runoff grid in each cell was expressed in that unit).

To take a break from so much theory, let's calculate a catchment area grid. Select the test.dgm grid in the *Elevation* field (I assume that you have preprocessed it), and press the *OK* button. Later we will study the remaining fields in the parameters window but, by now, focus your attention on the just created grid, which should look like this.



I know what you are thinking. Not very impressive, isn't it? It is almost completely white! There's much more information in the grid than what you can see, but the way this values are distributed results in such an "empty" appearance. For a more eye–catching representation, try a logarithmic scaling mode (in case you do not remember, this can be done using the settings window of the grid). The grid should now look like the one below.



Although there are differences between the catchment area grids created using one or another flow routing algorithm, their appearance will always be like that, since those differences are quite subtle and normallly can't be visually appreciated.

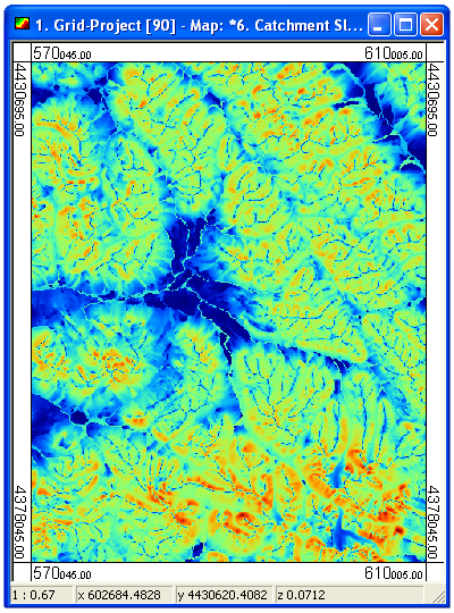
Going back to the parameters window, we find many other grids as output. Set them all to — *CREATE NEW* — so we can have a look at them and explain a couple of things about their meaning.

As it has been said, depending on which module you execute, you will be able to generate more or less new grids. Since we are using the *Parallel Processing* module, which is the richest

one, I will explain all of these grids, but be aware that not all of them can be generated with other modules. However, the first three (catchment area, catchment slope and catchment height) are always available.

Under the *Options* node you can find several fields regarding how all these grids are calculated. The flow routing algorithm to use can be selected in the *Method* field, and depending on which module you are running, you will find different options. The remaining fields contain parameters used by some of the available methods, but a deeper knowledge of them is needed to understand the meaning of this parameters (and even more to use them and adjust them to your needs) so I will not give further explanations about them. If you are interested in this subject, read the provided references and you will surely find what you are looking for.

The first grid apart from the catchment area one is the catchment slope grid.

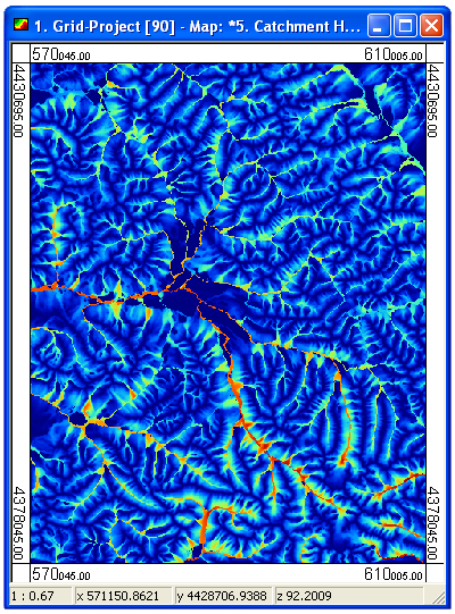


In it, the value of each cell represents the average slope of all the cells that drain to that one.

Slopes are represented in radians (the Z factor equals 1 in this grid), so take care if you are going to use this grid along with the slope grid obtained from the *Local Morphometry* module.

The last one of this three grids is the catchment height grid. Its values represent the average height of the upslope cells over each one. It is calculated as the average height of all upslope cells minus the height of the considered cell.

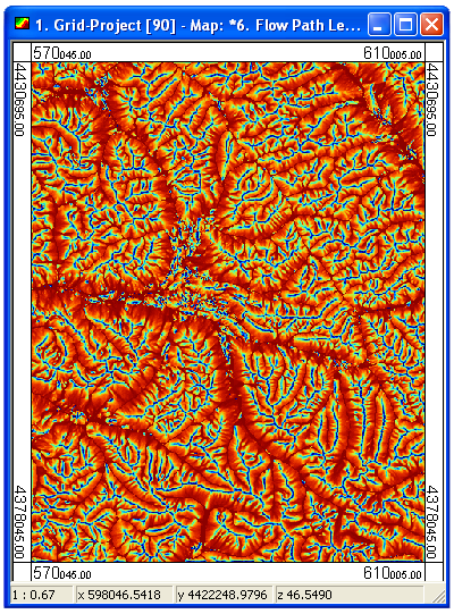
Here you have the catchment height grid created from the test.dgm grid.



Among those grids not available in all modules, we find the catchment aspect grid, whose values represent an averaged aspect of all the cells upslope each one.

This values are also expressed in radians, not degrees. I have changed the color palette, so the result you will have will look different.

The last grid that can be created is the flow path length grid.

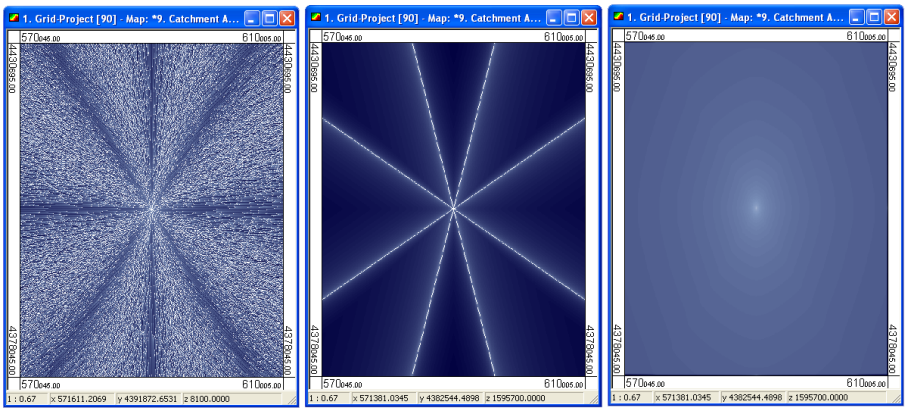


Its values are expressed in the same units as the grid cell size.

All these grids also depend on the flow routing algorithm, but that difference is not significant and it cannot be said that the more precise algorithms yield more precise result for this parameters.

Now that you know how to calculate catchment area grids, the differences between flow algorithms can be seen calculating a catchment area grid for a same DEM using all these algorithms. Instead of using the complex test.dgm grid, we will use a simpler mathematical semi–spherical surface, which can be created using the *Grid Function* module. Set the xmin and ymin values to -1 and the xmax and ymax values to 1 and use the following formula in the *Formula* field: `1-(x*x+y*y)`

Now, calculate catchment area grids using this grid as input and some of the available flow routing algorithms, and you will see clear differences between them. Below you can see, from left to right, the D8, Rho8 and MFD grids. Try yourself the other ones.

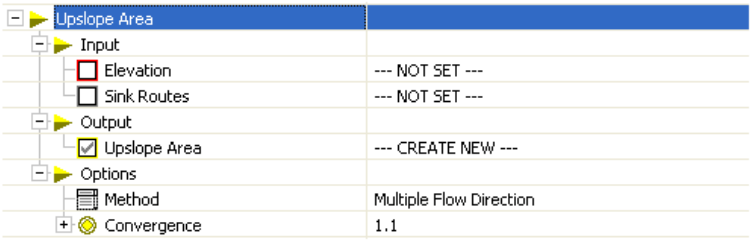


9.7 Calculating upslope and downslope areas

The value in each cell of the catchment area grid indicates the area upslope that cell. However, it is also interesting to know, not only the extension of this area, but also *where* this area is located, that is, the shape of the associated basin. Downslope area (its location) can be calculated as well, indicating through which cells the flow will go from a selected one.

All those results depend on the selected cell, and in this section we will see a new kind of module, an interactive one, that will appear in all those calculations that need the introduction of a cell as input. Instead of using the parameters window, the user can interact with the grid and use the grid view window to make this selection.

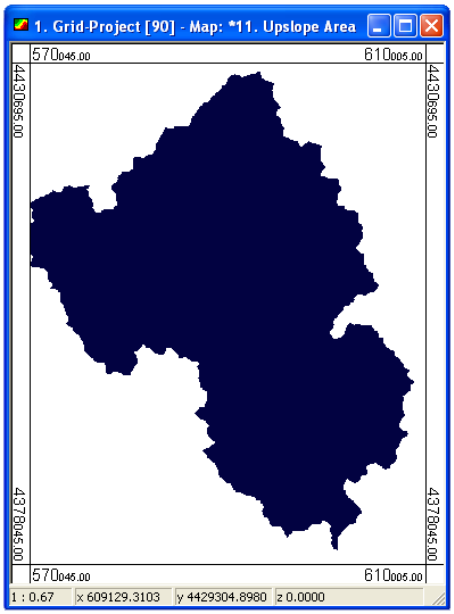
Select the *Flow Accumulation/Upslope Area*.



The parameters window looks quite similar to the one in the catchment area modules, but with less fields. No *Weight* field is found in here, and only an output grid is generated. Flow routing algorithms are also used, and you can select the one you prefer in the *Method* field. Some algorithms such as the flow tracing one are not suitable for this kind of calculation, so only some of them can be found in here.

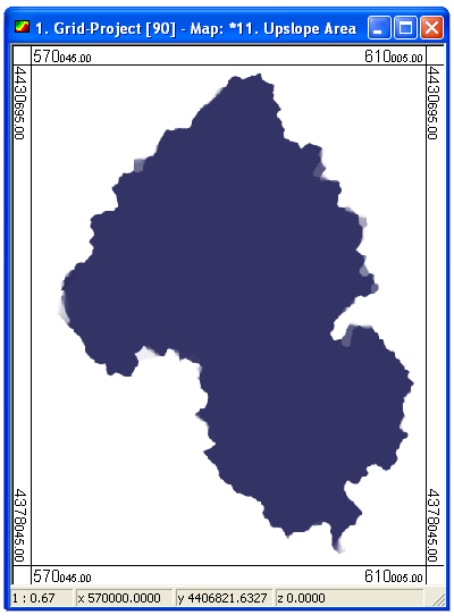
Press the *OK* button. Nothing happens (a grid has been created in the project window, but its thumbnail is completely blue). SAGA is waiting for a last input, but this time you have to enter it not with the keyboard but with your mouse. You can go to any grid within the active project and just click on the cell you want to select. In this case, it is more interesting to select a cell with a high catchment area value, so a bigger basin is calculated. Therefore, the catchment area grid should be used to select a cell.

Once you select it, SAGA will start calculating and the new grid will be created. Depending on the cell you pick up, you will have a different basin, but the resulting grid will look like this.



Probably you have noticed how similar it is to the bitmask grid that we used at the end of the last chapter. That is because I chose the D8 algorithm, and the flow from a cell can flow into the basin or not. Using other methods, since they consider bidimensional flows, a cell in the border of the basin can give its flow to several cells, some of them in the basin and some

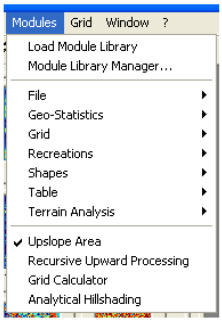
outside it. Its a subtle difference, but it can be easily appreciated if you compare the above image with the following one calculated using a Multiple Flow Direction algorithm.



This should help you to understand the differences that arise from using one algorithm or another.

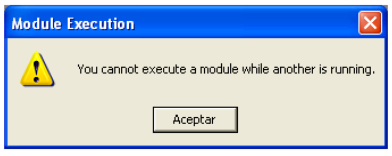
In the case of a "normal" module, the module is executed when the *OK* key is pressed and, once the results have been generated, the modules ends it execution and you can use other modules or do whatever you want without restrictions. However, interactive modules do not finish their execution once the results have been created.

If you go to the *Modules* menu, you will see at its bottom part the name of the current module, marked with a tick.

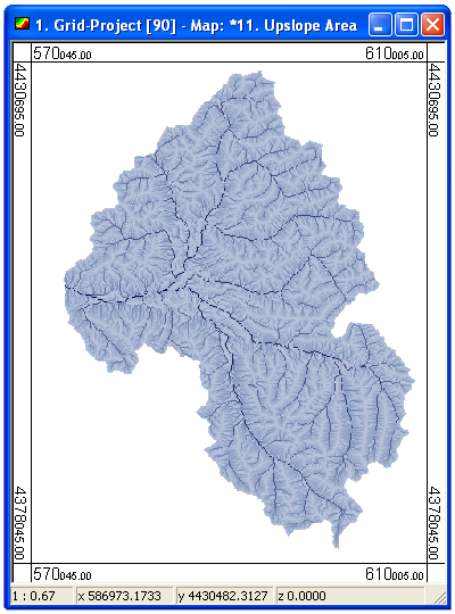


That indicates that the module is still active and, since it is an interactive module, waiting for a new input. If you select any other cell in the grid view window, all the results (in this case a new basin) will be calculated again, overwriting the already existent ones.

If you now try to execute another module, SAGA will not allow you do it, since two modules cannot be executed simultaneously.



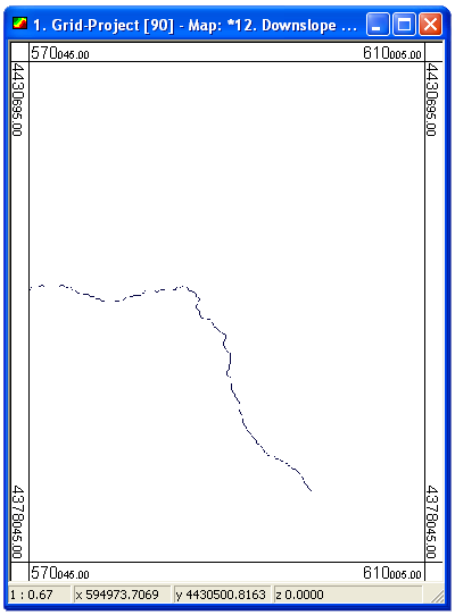
To finish module execution, go to the *Modules* menu and click on the menu item with the tick. SAGA not only removes that tick, indicating that now the module is not running, but also perform some final calculations. In the case of the *Upslope Area*, the basin seems to have vanished. SAGA has substituted the values of cells inside the basin with catchment area values, and all the outside cells have been assigned a zero value. If you set the scaling mode to *Logarithmic*, you will see the following grid.



Once you know how to use this module, it will not be difficult for you to understand how the *Downslope Area* works. Its parameters window is identical to the one in the *Upslope Area* method, and after you close it pressing the *OK* button you will have to pick up a cell in the grid view window. This time however, SAGA will assign a non–zero value to all the cells downslope the selected one, and a zero value to the remaining ones.

Notice that, unlike in the *Upslope Area* module, all flow routing algorithms are available in the *Method* field.

The resulting grid will look like this:

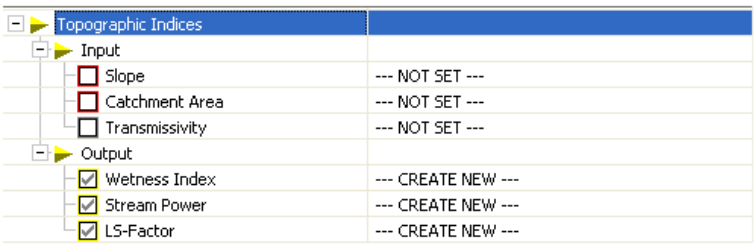


If you remember how to create profiles, this is rather similar (here a new grid is created, there a profile was calculated and the flow path was drawn onto the DEM grid). The flow path calculated to create a profile is calculated using the D8 algorithm.

9.8 Some hydrological indices

Catchment area grids can be used to calculate some very interesting indices, which will supply useful information about the hydrological characteristics of each cell. This indices are calculated using rather simple formulas that involve catchment area and slope as their main parameters, and that could be easily evaluated using the grid calculator. However, SAGA provides a ready–to–use module that will ease this task, avoiding the introduction of any formula.

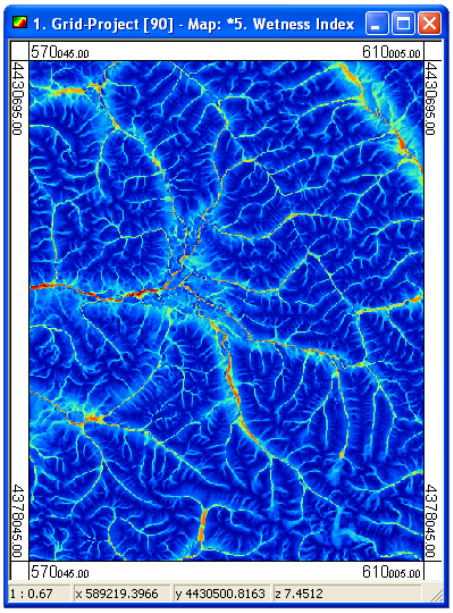
To start this module, select the *Indices/Topographic Indices* menu item.



As expected, slope and catchment area grids are required as input, and a third optional grid appears also under the *Input* node. This grid should contain transmissivity values to be used for the first output grid, *Topographic Index*. It is extremely rare to have information about transmissivity, so you will probably never use this field. If no grid is selected, a constant transmissivity is assumed.

Select the grids in the *Slope* and *Catchment Area* fields and press the *OK* button. You will get three new grids, which will be explained next.

The first one is the wetness index index grid.



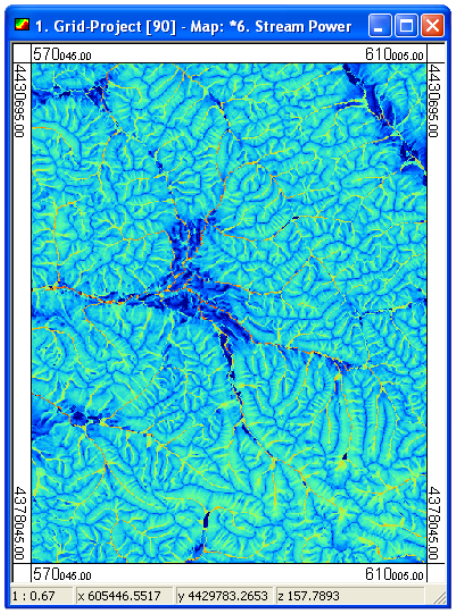
The wetness index (also known as *Topographic Index* is calculated using the following expression.

$$W = \frac{a}{\ln S} \tag{9.1}$$

where a is specific catchment area,
S is slope
and T_0 is soil transmissivity.

The wetness index comes from the TOPMODEL (Beven & Kirby, 198) hydrological model, and is related with soil moisture. It indicates the tendency of a cell to produce runoff, since areas with high moisture are more prone to become saturated. The higher the value of this index in a cell, the higher soil moisture that can be found in it.

Using slope and catchment area another index can be calculated: the stream power index.

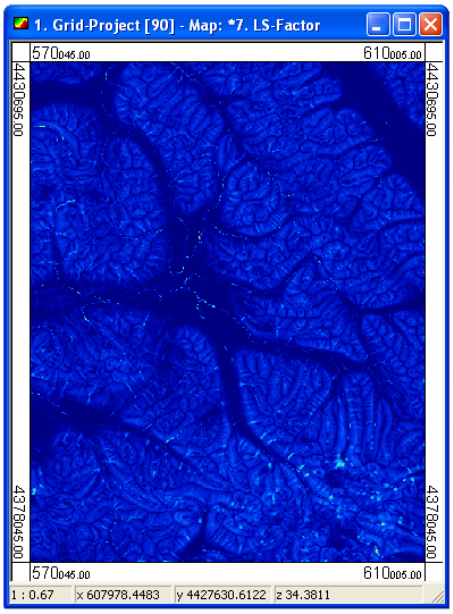


This index is related to erosion processes, constituting an indicator of the capabilities of a flow to generate net erosion. It is defined by the following equation.

$$\text{SPI} = aS \tag{9.2}$$

Another erosion related parameter is the LS factor, which is used in the well–known Universal Soil Loss Equation (USLE). However, the original equation for the LS factor — which used slope and slope length as main parameters —, has been substituted by a new one which uses catchment area as its main parameter, thus adapting it to the capabilities associated with DEMs.

An LS factor grid looks like this.



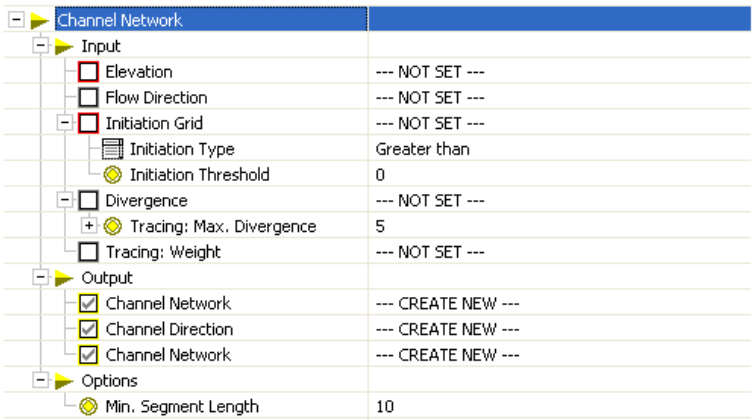
9.9 Defining channel networks

We have already use catchment area grids to obtain new grids which contained some other continuous parameters. However, one of the most important (if not the most) task that can be accomplished using catchment area information is the extraction of channel networks, a task completely different to what we have already seen.

Channels are usually located in areas through which a great amount of water flows, so this water defines the channel itself and modelates its shape. Therefore, it is quite logical to try to extract those channels from catchment area grids, which indicate the amount of cells located upslope each one and, consequently, can be used to evaluate the amount of water that will flow through it coming from all those upslope cells.

There are many different ways of using catchment area grids for this purpose, some of them involving additional grids, and each one of them has its pros and cons — as it happened with flow routing algorithms. They are not going to be explained in this book (once again, references for further study are given), but we will see how SAGA features a very flexible module that can be easily adapted to any of those methods.

To open it, select the *Channels/Channel Network* to get to the following parameters window.



Two grids are compulsory: The elevation grid that will be used to route the flow and trace the channels, and an initiation grid that will supply the information about where this flow has to be routed to create those channels. As usual, a flow direction grid can be used if the elevation grid is not preprocessed, so as to help SAGA route the flow over closed depressions.

The other key grid that has to be introduced is the *Inititation Grid*. SAGA will choose as initiation cells (cells from where the flow will be routed to trace channels) those cell in the initiation grid that fulfill a particular condition.

The differences between all the channel tracing methods lay in the characteristics of the initiation grid. Depending on the underlaying theoretical ideas of the method, a different initiation grid should be used. The most usual initiation grid to use is the catchment area grid, and this is what we will use in here. Using this grid leads to some undesired results such as a constant drainage density of the defined channel network, something that actually does not happen with "real" channel networks. Some alternatives have been proposed to overcome this drawback, such as using a combination of slope and catchment area as initiation parameter, but they are not frequently used, so I will stick to the classical catchment area–based method.

To define the condition under which a channel is started, you have three alternatives: the cell has a value lower than a threshold, the cell has a value that equals the threshold, or the cell has a value greater than the threshold. This threshold must be introduced in the *Initiation Threshold* field and should be in the same units as the initiation grid, so in our case it will be expressed in square meters.

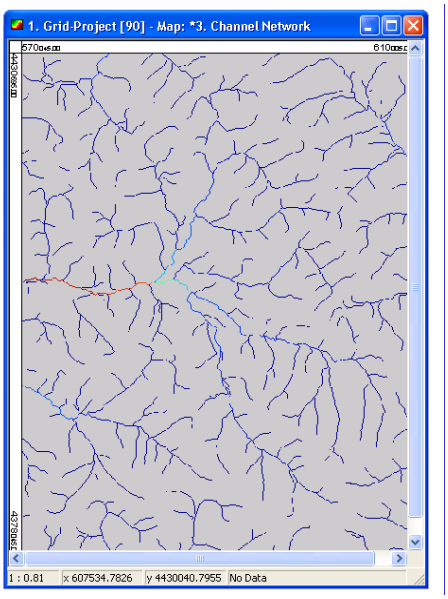
Set the *Initiation Type* to *Greater than*, so all cells with a catchment area greater than the threshold will be considered as part of a channel. For the *Initiation Threshold*, try a value of 1000000 (six zeros). The higher you set the threshold, the less cells that will be found in the initiation grid which fulfill the initiation condition, so the less channels that will be defined. There are a few different methods to choose the better threshold (some of them can be found in the references provided for further study), but the easiest one is just trying to make the defined channel network look as similar as possible to the "real" one.

To avoid very short channels segments being created, you can define a minimum length (in cells) using the *Min. Segment Length* field.

A couple of fields can be found under the *Divergence* node. They are related with flow tracing algorithms and a good knowledge of them is required to understand the meaning of these parameters and how to use it, so once again I will leave them unexplained. Advanced users will surely have no difficulties learning how to use these parameters themselves.

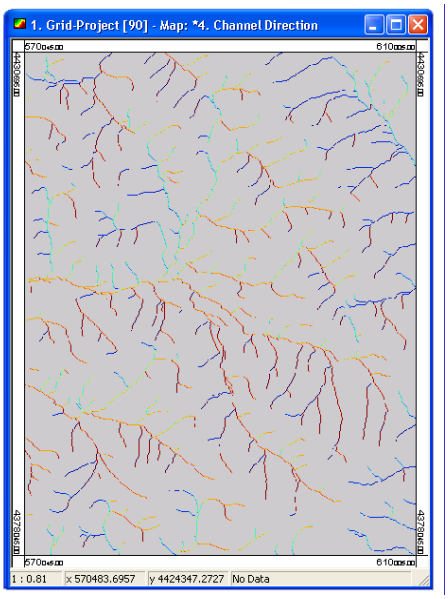
Finally, let's have a look at the outputs that will be produced. Since they are all by default set to — *CREATE NEW* —, just press the *OK* button and three new elements will be created.

The first one is the channel network itself, which should look like this.



It is quite similar to the testchan.dgm grid, but with a little difference: the drainage density is higher and this grid contains more channels than the testchan.dgm grid. This is due to the different threshold that has been used to calculate both grid. As it happened with the testchan.dgm grid, the values in the cells indicate the order or the segment that flows across each cell. Since more channels have been defined in this new grid, the maximum order found in this grid is greater than the one found in the testchan.dgm grid.

The second grid that has been created looks quite similar to the first one.



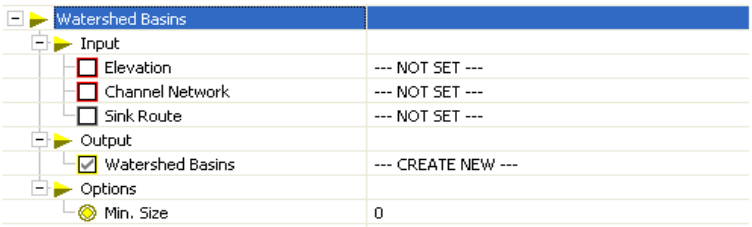
Here, the same channels are defined, but the cells values indicate in this case the flow direction of each cell. These directions are coded from 1 to 8, counting clockwise from northeast as follows:

7	8	1
6	X	2
5	4	3

The third output element created by this module is not a grid but a vector layer. Click on the *Shapes* tab in the project window and you will see a vector layer called *Channel Network*. It contains the same channel networks as the two previous grids, but in vector format.

9.10 Basins

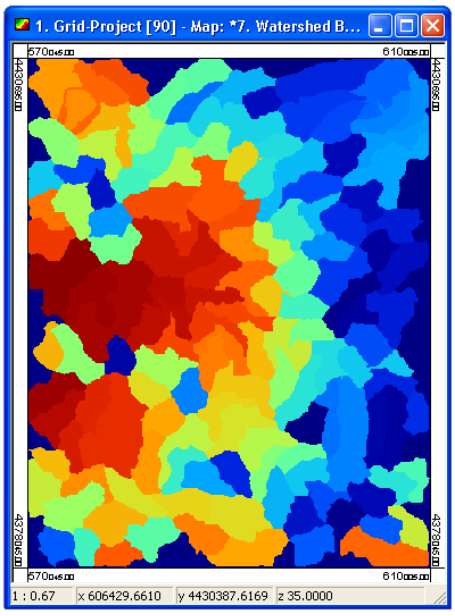
Each segment from the just created channel network has an associated basin (which corresponds to the catchment area of the lower point of the segment minus the basins associated to other segment located upslope). SAGA includes a module that will allow you to create a new grid containing all this basins. To run it, select the *Channels/Watershed Basins* menu item



The parameters window is pretty straightforward. Select a DEM (and a sink route in case it is needed) and then the channel network you want to use. All the cells that represent an intersection between different channel segments will be used as outlets of the basins you are about to create.

As with channel networks, you can also avoid creating very small basins, introducing a minimum size (in cells) in the *Min. Size* field.

Press the *OK* and you will get a grid like this:



As you can see, this is a grid containing discrete information. Cells of a same basin have the same value, which correspond to an ID number given by SAGA to each one of them.

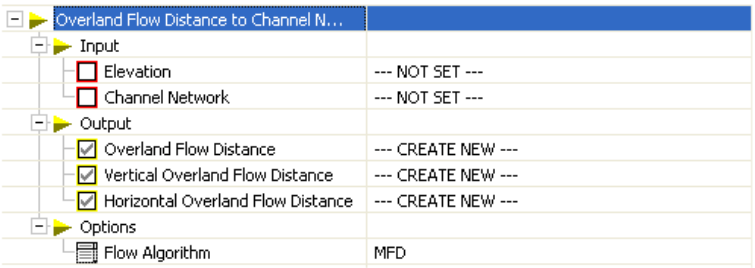
To adjust the number and size of basins to your needs, use different channel network grids. You can have a more detailed channel network grid to represent channels, but then create one using a greater threshold so it can be used to generate a basins grid with a lower number of basins (easier to use if you plan to introduce its information in a hydrological model).

Although it is quite interesting to have this information about basins in a vector format (polygons in this case), the *Watershed Basins* module simply creates a grid, leaving it up to you to convert its data into vector format or not. In a few chapters we will see how to create a vector layer from this grid, so you can use it along with the channel network layer.

9.11 Distances to channel network

Water behaves differently in channels than in other areas, and different equations can be used to define the flow inside and outside channels. Therefore, the distance between a cell and the closest channel is an interesting parameter than can provide some additional information about the hydrological characteristics of all those cells within a basin.

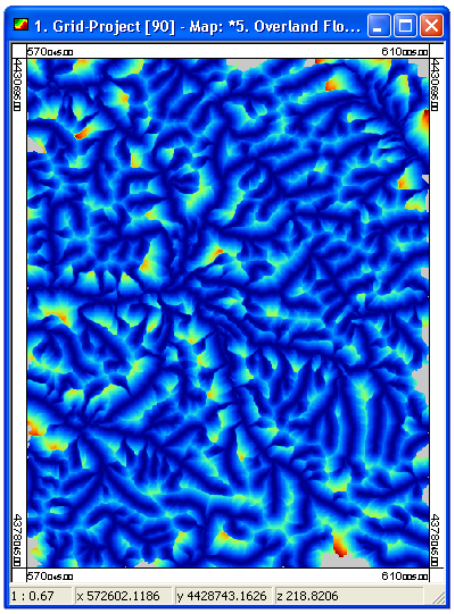
The modules included in SAGA feature two different approaches to calculate distances from non–channel cells to channels cells. The first (and the most simple) one can be reached through the *Channels/Overland Flow Distances to Channel Network*.



Two grids are needed as input. A DEM and a grid containing channels. This grid must have the same characteristics as the channel network grid defined by the *Channel Network* module, which means that all channel cells must have a valid value, while those cells outside the channel must contain a no data value. Values in channel cells are irrelevant here, so both the channel direction grid and the channel network grid can be used. Also, any channel network grid you create can be used, as long as it adhere to the previously introduced restrictions.

Three output grids are generated: one containing the overland flow distance and two more ones which contain the vertical and horizontal components of this distance. These distances are all expressed in the same units as the heights and cells size values from the DEM grid.

An overland flow grid should look like the one depicted below.



These overland flow distances are calculated not as euclidean distances, but taking into account the real movement of water from cell to cell. For that reason, a flow routing algorithm must be used, and it can be selected in the *Flow Algorithm* field. Only D8 and MFD algorithms are available.

Along with this, the *Vertical Distance to Channel Network* module implements a different idea, not based on flow distance (that means that no flow routing algorithms are used) but on vertical distance between cell elevations and the elevations calculated for the channel network in that cell. The process is as follows:

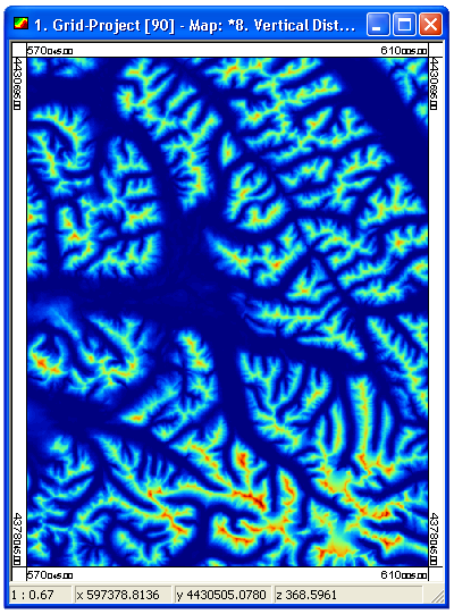
- The elevation of the channel network in a channel cell is simply the elevation of that cell in the DEM. For cells outside the channel network a cell elevation can be interpolated using the elevation values of channel cells. Doing this, a grid with channel network base level elevations can be calculated.

- Subtracting this grid from the DEM generates a new grid which contains the difference between cell elevation and the elevation of channel in that cell. Channel cells will get a zero value, while non channel cells will get a different one.

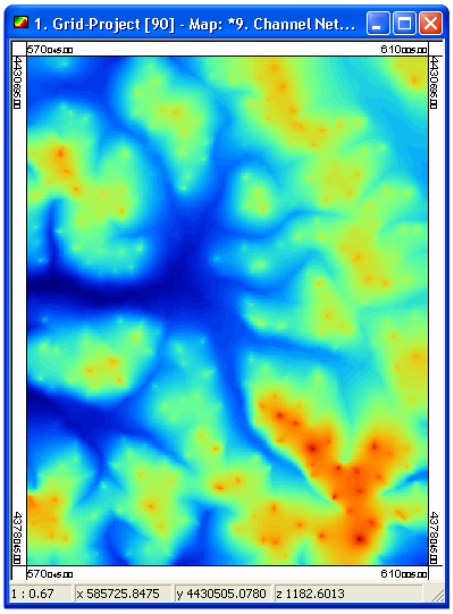
In other words, non channel cells will be assigned a value which represent the elevation difference between those cells and the channel that flows through them, in case it existed.

In the parameters window you will find the two same inputs as in the previous module: a DEM and a channel network grid. A *Tension* field can be found at the bottom of the parameter list, but I recommend not to modify it (the meaning of the results of this module is rather fuzzy itself, so trying to alter those parameters is, in the best case, something that most users will probably should never do).

The resulting grid will should like this.



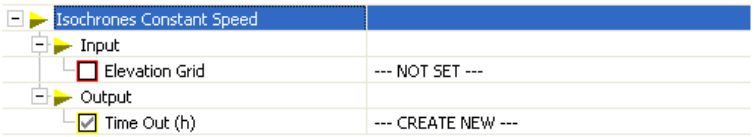
SAGA can also give as output the grid with channel network base levels for all cells, which will look like the one shown below these lines.



9.12 Calculating time to outlet. Isochrones

A very interesting parameter that can be calculated using the information in a DEM (particularly, using flow directions between cells) is the time that it takes to get out of a basin from each one of its interior cells. Having this information, not only the amount of runoff that will flow through the outlet cell can be known, but also how this runoff will be distributed in time.

Two modules are available for accomplishing this tasks, one of them being rather more complex than the other. The more complex the method is, the larger number of input parameters it will need. We will start with the simpler one, which can be run selecting the *Hydrology/Isochrones Constant Speed* menu item.



Using this module, a fixed speed is calculated from the characteristics of the basin, and water is assumed to flow at this speed through all the cells. Using flow directions, the flow length from each cell to the outlet can be calculated, and then converted into time using that speed.

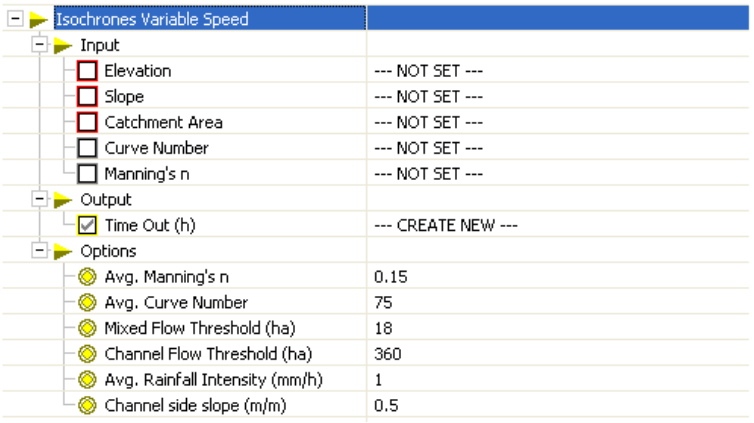
This module, like the other isochrones one, is an interactive one, so after you close the parameters window you have to select an outlet cell clicking on the grid view window. Once you select the cell, the selected basin will appear in the new grid, but until you stop the module it will not be represented with the right color pallete. The resulting grid will be like the one we already used to create a mask in the previous chapter.

Times in the isochrones grid are expressed in hours.

This grid has a great hydrological meaning. For example, the frequency histogram can be used to calculate an unit hydrograph of the basin, surpassing the accuracy of other simpler alternatives such as the well–known triangular unit hydrograph. This will not be a synthetic hydrograph, but a "real" one, calculated using information from the basin itself and some simple physical laws about water flow.

However, it is not very realistic to consider that water moves at the same speed through all the cells (time area usually under–estimated using this method). Those simple physical laws can be replaced with more elaborated sets of equations, which will assign a different flow speed to each cell according to its properties (slope, amount of water that is expected to flow through the cell, flow rugosity...).

To use this more complex method, select the *Hydrology/Isochrones Variable Speed* menu item.



As you can see, the parameters window is much more populated than the last one, containing not only grid inputs, but also several numerical parameters under the *Options* node. I will explain them all in detail.

Three grids are compulsory: elevation, slope and catchment area. You should need no further explanation about them. Below these, you can find two new optional grids: Curve Number and Manning's n number.

The Curve Number grid is used to estimate runoff from rainfall, and, with the information given in the last chapter, you should be able to create one yourself. Manning's n number defines the flow rugosity of each cell, and is used to calculate flow speed. If you have no gridded information about these parameters (CN data is not hard to find, but Manning's n

data is), you can use a constant value for all cells. To do that, leave the *Curve Number* (or *Manning's n*) field set to — *NO SET* — and introduce the value to use in the *Avg. Curve Number* (or *Avg. Manning's n*) field. This will have the same effect as selecting a grid with a constant value equal to the one introduced in the corresponding average value field.

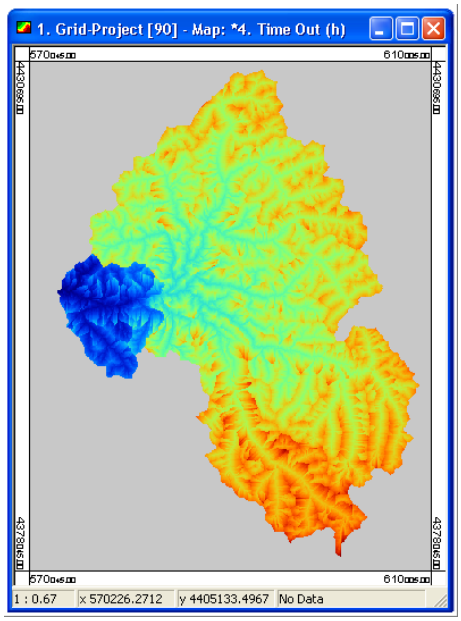
Rainfall intensity, which is assumed to be constant all over the watershed, must be introduced in the *Avg. Rainfall Intensity*. Larger values will cause flow to be faster and, consequently, times in the grid to be shorter.

The remaining values are related with the three types of flow that are distinguished when using this module: overland flow, mixed flow and channel flow. The boundaries of each one are defined using catchment area values, and must be introduced in the *Mixed Flow Threshold* and *Channel Flow Threshold* fields. Unlike in the *Channel Network* module, where threshold was expressed in the same units as cell size (squared, of course), here it is expressed in hectares. However, the default values are suitable for most cases, and changing them should only be done under some "rare" circumstances, when additional information is known (something that seldom happens...).

Mixed and channel flows are assumed to go through a triangular channel. The slope of both sides of this triangle can be adjusted modifying the values in the *Channel side Slope* field.

A minimum speed can be set in the *Minimum Flow Speed* field to avoid very low speeds in the case of almost flat cells or low runoff, specially in those cells with overland flow.

As shown under these lines, the grid generated from this module is quite different to the one created with the *Isochrones Constant Speed* one, and looks less "uniform", since each cell has a different flow speed.



9.13 Other hydrological parameters

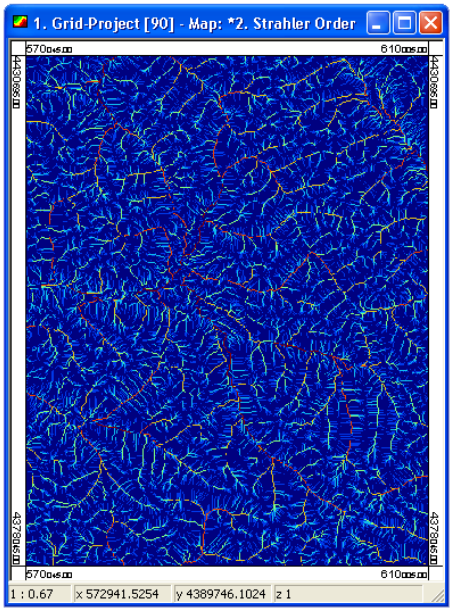
Apart from the isochrones modules, the Hydrology.mlb library contains some other minor hydrological modules. Here, we will just have a look at one of them, but I invite you to use the other ones and try to understand their meaning and how they work (which is not difficult at all, believe me...).

Among this modules, the *Strahler Order* one generates grids in which each cell is assigned its Strahler order. Strahler orders are just another way to define the hierarchy that exists

between different river segments. However, instead of just analyzing those cells through which a "real" river flows (that is, those calculated using the *Channel Network* module, it considers every single cell as part of a river. This will create a grid in which all cells have an assigned value, that can be later used in the *Channel Network* module as the threshold grid.

To create this grid, select the *Strahler Order* menu item. As input, you just have to select the elevation grid. This has to be an already preprocessed DEM, since flow algorithms will use it.

The resulting grid will look like this.



It looks quite similar to the catchment area grid, but here a linear scale is used instead of a logarithmic one. Also, values in this grid range just from 1 to 8 and their distribution is rather different. Since, as it has been said, this grid can be used as a threshold grid in the *Channel Network* module, that means that a channel network can be defined from a DEM without needing a catchment area grid!! Try to define a channel network using this Strahler order grid and compare the results with the ones obtained earlier in this chapter. This should help you better understand how the *Channel Network* module works.

9.14 Lighting

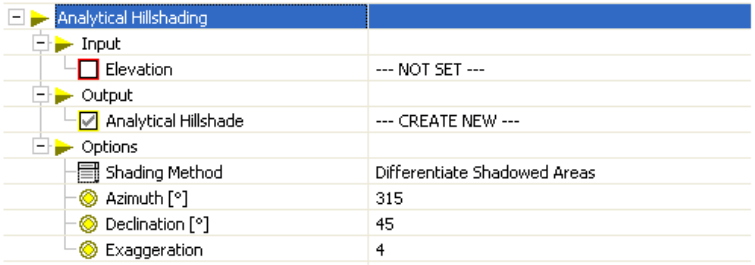
Terrain relief affects the way that shadows are casted over different cells, thus modifying its appearance, but also altering some very important parameters such as the amount of energy that is received from the sun in a cell. With the sole aid of a DEM and a couple of basic astronomical ideas, we can create a shaded relief representation of a grid (which, as you already know, can be used to shade any other grid of that same area) or calculate the radiation income in all of the cells of the grid, among other things.

Tasks like these can be accomplished using the modules under the *Terrain Analysis/Lighting* modules, which we will study in this section.

The first result we are going to obtain is one with which you should have already become acquainted: a shaded relief grid. A shaded relief grid is a pseudo 3–Dimensional representation of a DEM, in which cell values reflect the amount of light that each cell receives. Representing this grid using a continuous bicolor color ramp gives as result a quite realistic (and very eye–

catching) depiction of the area described by the DEM, as seen from above (imagine that you are flying on a plane just above this area).

The *Analytical Hillshade* module is the one that must be used to create a shaded relief grid.



As input, just the DEM itself is needed. Since we are dealing with lighting algorithms, a light source must be defined, which can be done modifying the values in the *Azimuth* and *Declination* fields. These are the polar coordinates of the light source (i.e. the sun).

Three methods are available.

- Simple
- Differentiate Shadowed Areas
- Ray Tracing

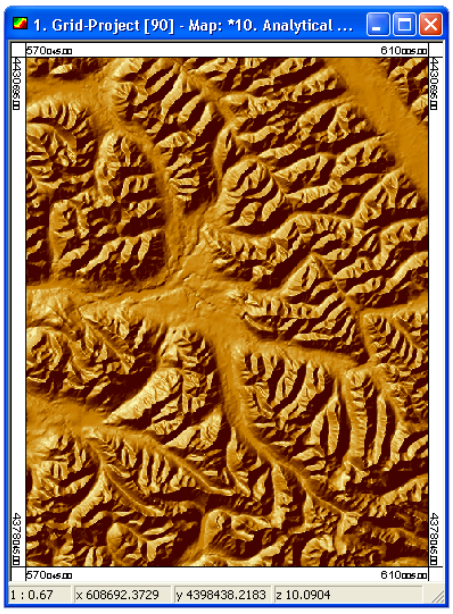
Try them to see which one you like the most. Since the result of this module is rather "visual", you should judge the convenience of them by the visual appeal of the grids they produce.

An exaggeration factor can be introduced in the *Exaggeration* field. It changes the elevations of cells, so slopes are also modified and, consequently, shadows get affected. The higher elevation value you choose, the darker that slope areas will look, in contrast with the brighter values of flat areas.

When I speak of "dark" and "bright", I means those cells that have high and low values, respectively. If you use a color pallete like the one set by default, higher values are assigned a dark color while low values area assigned a bright color. Try changing it to get a different image. Inverting the color pallete will make higher areas look sunken under low areas, which will look more elevated, so the relief itself gets completely inverted.

If the *Ray tracing* method is used, the exaggeration value will not be used.

Although a shaded relief grid can be used to shade another grid, it is itself a very interesting result, as can be seen below these lines.



The values in the shaded relief grid are angles expressed in radians, but, once again, as it happened in the case of slope and aspect grids, a convenient Z factor is used to change those values from radians to degrees.

For a more *quantitative* result, we will see how to calculate solar radiation income using another lighting module. Select the *Lighting/Solar Radiation Sums* menu item.

Solar Radiation Sums	
Input	
Elevation	--- NOT SET ---
Output	
Solar Radiation	--- CREATE NEW ---
Update View	true
Unit	kWh/m²
Duration of Insolation	--- CREATE NEW ---
Options	
Solar Constant	1367
Transmittance of Atmosphere [%]	60
Latitude	53.5
Daily Time Resolution	
Time Span [hours]	
Minimum	0
Maximum	24
Time Step [hours]	1
Simulation Time	Single Day
Single Day	
Day	21
Month	March
Range of Days	
Time Span [day of year]	
Minimum	1
Maximum	31
Time Step [number of days]	5

The parameters window is slightly more complex that the ones we have already seen in this section, since not only the DEM is needed, but also some additional information about the characteristics of the radiation and the time period to evaluate.

Before we start analyzing each field in the parameters window, let's have a look at how the resulting grid is calculated. The algorithm can be divided into the following main steps:

- A time range is defined.

- This time range is divided into single days.
- Several equally spaced "instants" are defined within each day.
- For each one of this "instants" (which implies a particular position of the sun), solar radiation is calculated for each cell in the grid, using the elevation information contained in the DEM.
- With all the individual solar radiation grids calculated, a daily radiation grid can be created.
- With all the daily radiation grids, the total radiation grid is calculated for the whole time range.

Now let's see how to make all the settings so SAGA can perform this calculations.

First of all, let's select the input and output grids. Nothing is new in the *Input* node, but we can see two options in the *Solar Radiation Sum* field under the *Output* one.

Since several grids are going to be calculated, you can choose to view them one after another and thus see the progression of the module (set the *Update View* field to *true*), or not (set it to *false*). This last option will shorten module execution time.

Select the units for the output grid in the *Units* field.

Along with the solar radiation sum grid, a duration of insolation grid is calculated. A larger duration of insolation is not necessary related to a larger radiation sum, since other parameters such as slope also have a significant influence on it.

Let's get into the *Options* node.

Use the *Solar Constant* and *Transmittance of Atmosphere(%)* fields to set the characteristics of the radiation that reaches the earth. Default values are a good choice for most cases, so you are likely to leave them unchanged.

You must set the latitude in which the DEM is located, using the *Latitude* field. This will affect the position of the sun in the sky within the selected time range.

A key element to define is the time resolution to use, that is, the spacing between those aforementioned "instants" in which each day is divided. The shorter time interval you choose, the more grids that will have to be calculated and thus, the more precise the result will be, but also the longer time the module will take to execute.

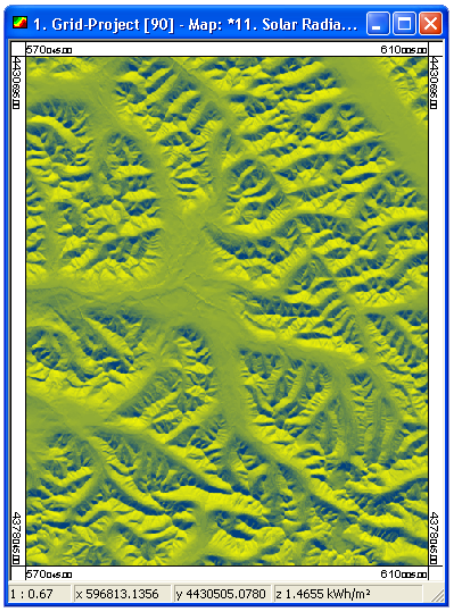
For the calculation of daily sums, you can define a time span in the *Time Span* field, so just a part of the day is analyzed.

Defining the time range is the last thing that has to be done. Three options are available:

- A single day.
- A range of days.
- A whole year.

For the two first ones, a corresponding node can be found in which additional information has to be entered. The fields under these nodes are all rather self–describing.

Once all the information has been provided to the module, press the *OK* button. When module execution finishes (might take a long time, depending on the settings you made), you will get a grid like this.



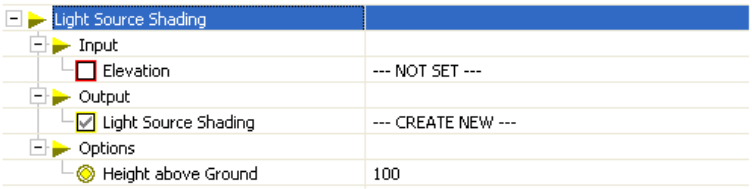
The last lighting module, which can be executed from the *Light Source Shading* menu item, is an interactive one. Instead of considering a light source situated in the sky (an defined by its azimuthal and elevation angles), it considers one located in a grid cell which must be defined clicking on the view grid much in the same way as in some of the hydrological modules we have already seen.

The module will create a grid indicating which cells receive the light from the source, and the magnitude of it, once again expressed as an angle.

What can this be used for? Imagine that you have to assess the visual impact of a building. Place the light source in the location of that building and you will see which cells receive the light from the building (that is, which cells *see* the building an are thus affected by its presence).

Another example: you want to place an antenna and do not know whether its signal will reach all that surrounding areas or some of them are *shadowed*. Run the *Light Source Shading* module and place the light source where the antenna is to be located. With the resulting grid, use the *Change Cell Values* to set a threshold over which cells can be considered shadowed (they do not "get" enough radio waves from the antenna), and you will get a new grid indicating those cells to which the antenna will give service.

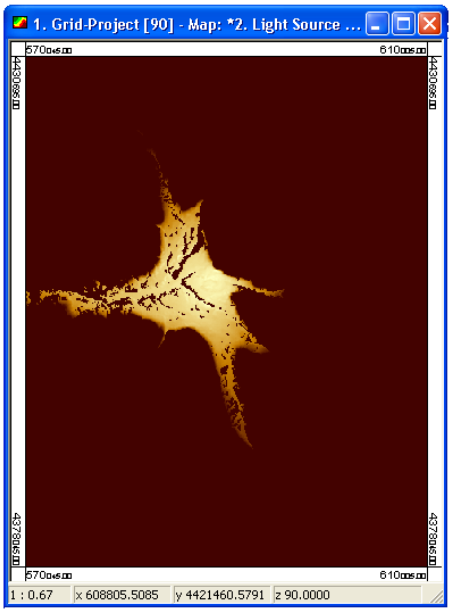
The parameters window of this module looks like this.



A simple DEM is needed as input, and just one grid is generated. The only option field allows you to set the height over terrain level of the source light (i.e. the height of the antenna in the previous example). If you set a very high value in this field, almost all cells will receive light, while only a few will (of course, depending on the characteristics of the terrain and the chosen cell) if a lower height is introduced.

The height of the source light must be in the same units as the elevation values of the DEM.

A typical grid obtained from this module is shown below.



All values in this grids are lower than 90, and indicate the angle between the light coming from the source light and the terrain. A value of 90º indicates a completely shadowed area. The lower the value, the more light the cell receives perpendicularly to its surface.

Chapter 10

Shapes Modules

10.1 Introduction

Though not as numerous as raster ones, several vector modules have been developed for its use in SAGA. Among them, some just deal with vector layers, while others combine vector and raster data.

Modules in this last group are probably the most important ones, since they allow to use both data format almost seamlessly and profit from the advantages of each one of them. Using this modules along with the ones introduced in chapter 7 should be enough to prepare all your data independently of its format and also obtain some interesting results.

At the end of this chapter you will see how SAGA vector capabilities, though rather limited, can be incredibly useful thanks to these modules.

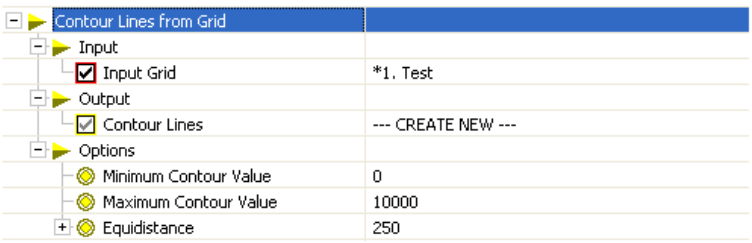
10.2 Combining raster and vector layers

We will start the chapter analyzing those modules developed to ease the simultaneous use of vector and raster data layers. Learning how to use these will allow you to, for example, convert between data types or enrich the information in a layer with that contained in another of a different type.

Not all the modules that work with vector and raster layers at the same time are included in this section (interpolation modules create grids from a points vector layer and are described in a separate chapter), but just those ones that can be found under the *Grid/Shapes* menu (and one under the *Grid/Gridding* menu).

10.2.1 Creating contour lines from a grid

Let's start with the first one. Select the *Contour Lines from Grid* menu item.



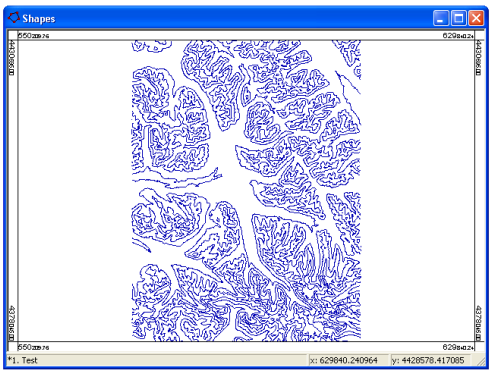
Before the introduction of DEMs and digital cartography, elevations were usually represented using isolines (i.e. lines along which a constant elevation value is found). These isolines

are still used today not only for elevations, but for many other parameters such as rainfall, temperature.... Of course, and although this is not the primary aim of SAGA, isolines are also used to describe mathematical surfaces and other mathematical concepts.

The *Contour Lines From Grid* module creates a vector layer containing isolines corresponding to a selected grid.

Select the input grid and the output vector layer in their corresponding fields. Select the range of values to consider to create the isolines and then introduce the elevation difference between lines in the *Equidistance* field. The lower equidistance, the more lines that will be created.

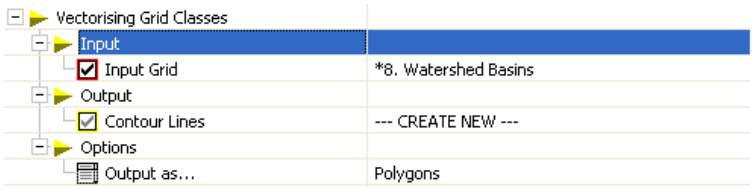
Here is the result for the test.dgm grid and an equidistance of 250 meters.



The inverse process, creating a grid from contour lines, can be accomplished using two different modules, one of which will be described in this same section.

10.2.2 Vectorising grid classes

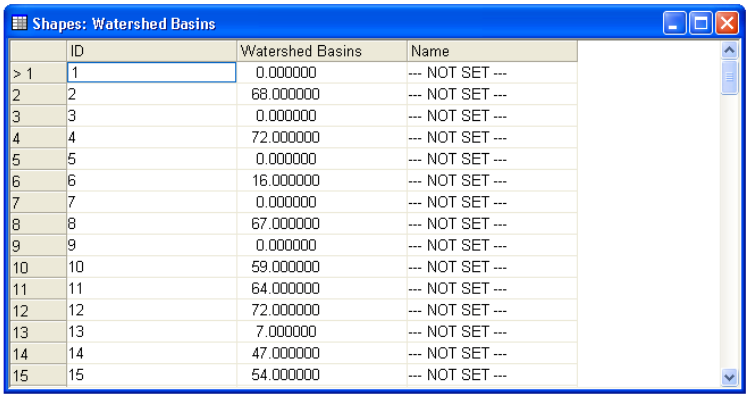
One of the most useful modules in the *Grid/Shapes* menu is the *Vectorising Grid Classes* one.



If you remember from the terrain analysis chapter, the *Channel Network* module created raster and vector layers containing the calculated channel network, but only a raster layer was created when defining watershed basins from that channel network. We will now see how to create a vector layer using that grid information.

Select the grid containing the basins in the *Input Grid* field. Set the *Output as...* field to *Polygons*. When you press the *Ok* button, SAGA will take all cells with the same value and create a polygon for each group of cells. The resulting shapes layer will be added to the shapes project window. I guess you can recognize it, since it is the same as the test2.shp file with which we worked in the *Working with shapes* chapter.

Have a look at the attributes table of the just created vector layer.



	ID	Watershed Basins	Name
> 1	1	0.000000	--- NOT SET ---
2	2	68.000000	--- NOT SET ---
3	3	0.000000	--- NOT SET ---
4	4	72.000000	--- NOT SET ---
5	5	0.000000	--- NOT SET ---
6	6	16.000000	--- NOT SET ---
7	7	0.000000	--- NOT SET ---
8	8	67.000000	--- NOT SET ---
9	9	0.000000	--- NOT SET ---
10	10	59.000000	--- NOT SET ---
11	11	64.000000	--- NOT SET ---
12	12	72.000000	--- NOT SET ---
13	13	7.000000	--- NOT SET ---
14	14	47.000000	--- NOT SET ---
15	15	54.000000	--- NOT SET ---

The common value that all the cell within a polygon share is added in a column with the same name as the source grid.

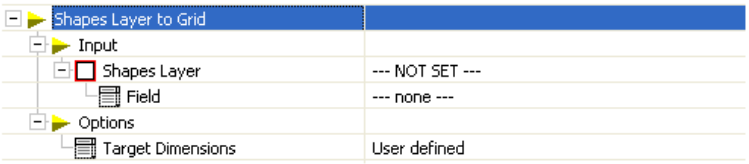
You will see several basins with a zero value. Those are the shapes that correspond to areas not into a basin (located in the boundaries of the grid), which are represented by several disjoint polygons.

Don't close this vector layer, we will use it for a future example.

10.2.3 Converting a vector layer into a grid

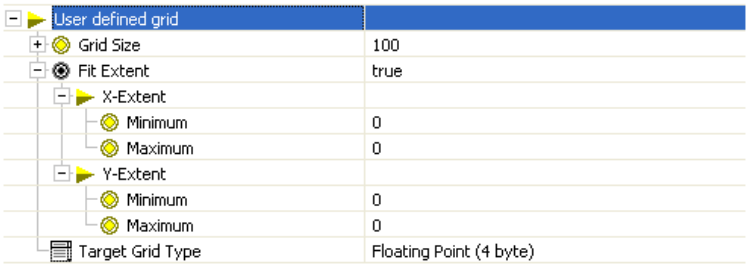
The inverse process to the previous one, converting a vector layer into a grid layer, can also be quite easily accomplished using another SAGA module.

Select the *Grid/Gridding/Shapes Layer to Grid* menu item.



Select the vector layer to convert in the *Shapes Layer* field. Since grids can only contain a single value for each cell, you must choose which field from the attributes table of the selected layer you want to use. Do it in the *Field* cell. Of course, you must select a numerical field, not a string one.

The usual alternatives for defining output grid dimensions are found in the *Target Dimensions* field. The *User defined* option will, however, take you to a different parameters window.



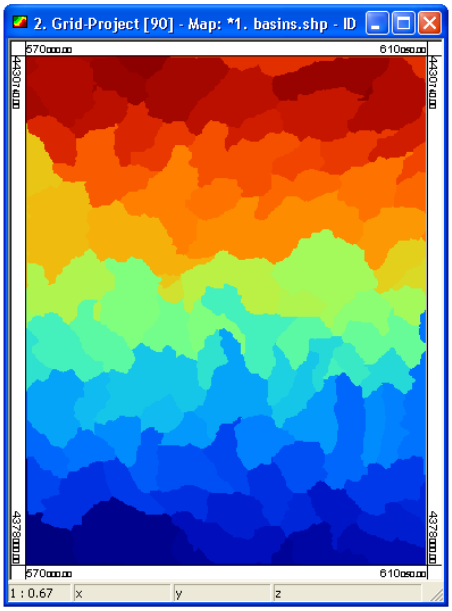
You must define the cell size in the *Cell Size* field, which must be in the same units as the source shapes layer.

To define the extension of the grid, you can introduce its boundaries (also in the same units as the shapes layer) in the four fields found under the *Fit Extent* field. If you set this

Fit Extent field to true, the four boundary values will be ignored and the new grid will have the smallest extension that fits to the one occupied by the shapes in the vector layer.

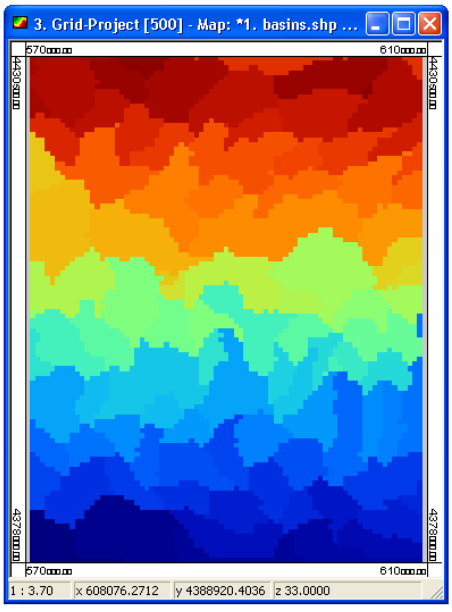
Lastly, you must set the data storage type of the new grid. You will found this field in all the parameters windows, independently of how you introduce the target grid dimensions.

Let's see an example. Convert the test2.shp vector layer containing basins into a grid using the *ID* field. You will get the following grid.



Why those colors? When basins are converted from raster to vector, the algorithm runs from the top to bottom of the grid, so the upper ones get a lower value. If you want a not so *straight* coloring, go to the settings window and create a random color pallete.

The above grid has a cell size equal to 90 meters, the same as the original test.dgm grid from which basins were calculated. Using a coarser resolution might cause the resulting grid to be rather worse, as can be seen in the following image, with a cell size of 500 meters.



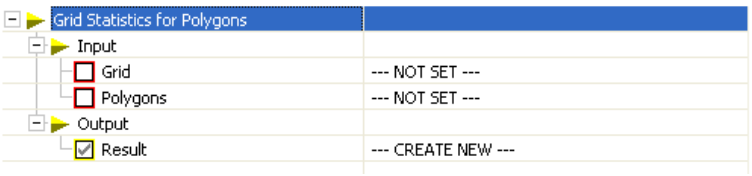
When converting vector features to raster you will always loose the inherent accuracy of them. Be sure to choose a cell size that keeps as much information as possible, while not resulting in overwhelmingly large grids. As with many other things, it's a matter of balance...

10.2.4 Retrieving grid information to enrich raster layers

The information contained in the attributes table of a raster layer can be enhanced adding more data to it from one or more grid representing the same geographical area. SAGA has several modules to do that, all of which will be described in this section.

In the case of polygons, the first thing we can do is to calculate average values. For example, it is very interesting to combine the just created basins layer with the Curve Number layer to get average CN value for all the subbasins. This can be used to calculate runoff for each one independently and then combine the results to get a more precise hydrological model than just using an average CN value for the whole basin.

Select the *Grid Statistics for Polygons* menu item.



Select the grid and the polygons layer to use and press the *OK* button.

You will see have a new vector layer identical to the basins one, but with an extended attributes table.

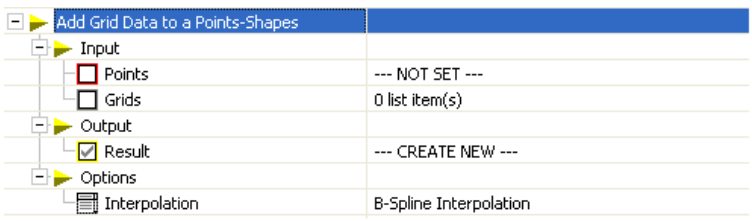
Shapes: basins

	Name	CELLS	MEAN	VARIANCE
> 1	--- NOT SET ---	1111	1644.895590	37689.380638
2	--- NOT SET ---	1328	1616.856175	27501.737597
3	--- NOT SET ---	268	1496.503731	21883.287300
4	--- NOT SET ---	6263	1492.656578	48812.878089
5	--- NOT SET ---	794	1491.755668	38178.434004
6	--- NOT SET ---	937	1296.421025	69567.305487
7	--- NOT SET ---	618	1697.773463	45959.502079
8	--- NOT SET ---	1625	1756.504154	57565.068729
9	--- NOT SET ---	928	1863.798491	50736.296679
10	--- NOT SET ---	2618	1662.955443	117564.183573
11	--- NOT SET ---	17	1418.058824	45475.467128
12	--- NOT SET ---	2511	1642.921187	84400.797896
13	--- NOT SET ---	1667	1428.113008	81957.241468
14	--- NOT SET ---	2	1102.500000	72.250000
15	--- NOT SET ---	2279	1854.212045	68695.360568

The number of cells that lie inside each polygon, the average value of those cells and its variance have been included in the attributes table.

You can also select the same layer as output and the old table will be replaced by the new one.

There are other ways of adding new information to a vector layer using one (or several) grids. If instead of a polygon layer you have a points layer, you can use the *Add Grid Data To a Points–Shapes* module



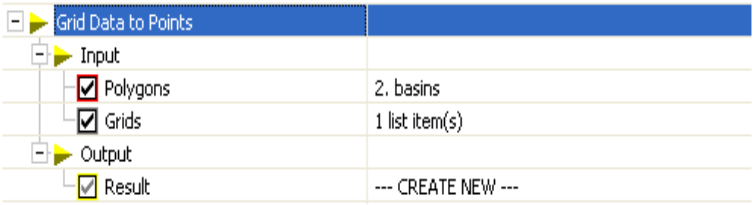
SAGA will take the coordinates of the point contained in the selected input layer and calculate the values at these locations in the chosen grids. Click on the *Grids* field to get to a multiple selection dialog where you can select the grids you want to use.

A new points layer will be calculated with as many new fields in its attributes table as grids were used.

Point coordinates might fall inside a grid cell or covering several of them, depending on the cell size of the grid. To calculate exact values at point locations, an interpolation algorithm is used. You can select it in the *Interpolation* field.

If the point falls outside the grid, a no–data value will be assigned.

Points layers can be created from grids without needing an already existing one. The *Grid Data to Points* creates a table that contains the coordinates of all the cells inside the polygons of a polygons layer, and each one of them is accompanied by the values of that cell in one or more grids.



This table can be used to create a points layer, as we will shortly see.

Introduce the polygon layer in the *Polygons* field (use the basins layer in this case). Again, select the grids from which you want to take the data using the *Grids* field.

Press the *OK* button and a new table will appear in the tables project window.

Using the test.dgm grid as input, the resulting table looks like this.

Table: --- UNNAMED ---

	ID	X	Y	test.dgm
> 1	1	570045.000000	4378045.000000	1423.000000
2	1	570135.000000	4378045.000000	1414.000000
3	1	570225.000000	4378045.000000	1392.000000
4	1	570315.000000	4378045.000000	1377.000000
5	1	570405.000000	4378045.000000	1360.000000
6	1	570495.000000	4378045.000000	1370.000000
7	1	570585.000000	4378045.000000	1403.000000
8	1	570675.000000	4378045.000000	1442.000000
9	1	570765.000000	4378045.000000	1466.000000
10	1	570855.000000	4378045.000000	1485.000000
11	1	570945.000000	4378045.000000	1502.000000
12	1	571035.000000	4378045.000000	1519.000000
13	1	571125.000000	4378045.000000	1516.000000
14	1	571215.000000	4378045.000000	1495.000000

Huge tables might result from this module, specially when using large grids, so don't get alarmed it module execution takes a long time.

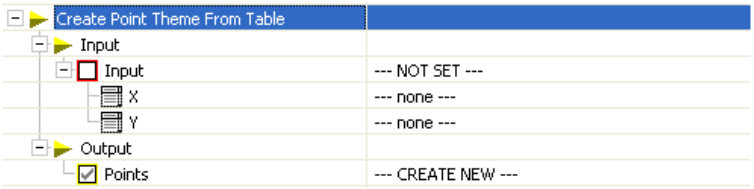
10.3 Creating a points layer from a table

You already know how to create a points layer or any other vector layer using the vector editing capabilities implemented in SAGA. Along with that, point layers can also be created from pure numerical data, without having to use the shapes view window.

What's the advantage of creating a layer this way? Well, first of all, it allows you to turn tables like the one we obtained from the *Grid Data to Points* module into vector layers. Second, you can create your own tables within SAGA. And third, if you need a more complex

(but somehow regular) point distribution, you can use a spreadsheet, then save the file, import it into SAGA and create the layer from it.

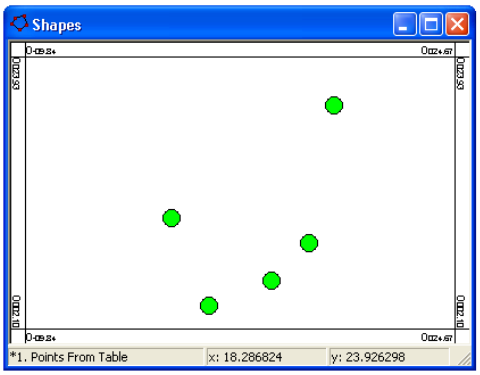
Select the *Create Point Theme From Table* menu item.



In the *Input* field, select the table where coordinates are stored. As you select it, the list in the *X* and *Y* fields will get populated with the names of the columns of that table. Select which one contains each coordinate.

Press the *OK* button and you will get a new points layer whose table is exactly like the source table, including even the coordinates columns. This is quite similar to importing a XYZ file, as we saw in the corresponding chapter about IO modules.

Try creating a points layer using the example table from the *Working with tables* chapter. You should get something like this:



We will later use this layer to create a grid using interpolation modules, so do not close it yet.

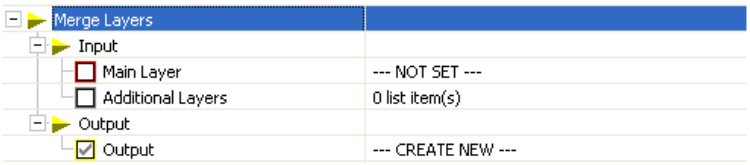
10.4 Merging vector layers

One of the most interesting features of vector layers is that you can merge them (of course, as long as they contain the same type of data) and create a unique one which includes all their data. This is not only useful for keeping the information in a more handy and compact manner, but also for many other reasons. For example, imagine that you have several different files containing countries defined like polygons, one file for each continent, and you plan to create a layout with all of them representing all the countries of the world. To color them, you are going to use the *population* field that can be found in all the associated attributes tables.

If you use the same color pallete from all the layers, the most populated country of each continent will get the same color, meaning that China and Germany will have the same color, despite the fact that the former has a population almost 15 times as bigger as the latter. You have to use the same color pallete and also the same min and max values for all the layers, but this is not possible, since coloring setting are made individually for each one. Therefore, you need to put them all in just one global layer.

Even when two layers have the same data type, it might not be a good idea to merge them. It is no sense merging a roads layer with a rivers layer, as their attributes table will be completely different. Only merge layers which represent the same kind of geographical elements.

To merge several layers, select them *Merge Layers* menu item.



There are two input fields: *Main layer* and *Additional Layer*. The layer you select in the first one will define how the table of the resulting layer will be. All the fields in its attributes table will appear in the attributes table of the output shapes layer, while those from the other layers to merge that are not found in the table of the main layer will be ignored. If the main layer contains a field not found in an additional layer, the entities of this layer will have no associated information for that field (the corresponding cells in the table will be empty)

After that, click on the *Additional Layers* field and you will get to a multiple selection dialog. Select all the layers that you want to merge with the main one.

Press the *OK* button to start module execution and you will get a new layer named *Merge*.

10.5 Joining tables

The attributes table of a vector layer might be extended using information stored in another different table. If they both share a common field, coincidences of that field can be used to join them. Let's see a little example. Have a look at the following tables.

City ID	City Name
1	Madrid
2	Barcelona
3	Sevilla
4	Zaragoza
5	Alicante

City Name	Population
Madrid	5086635
Barcelona	3765994
Sevilla	1180197
Zaragoza	638535
Alicante	380357

Using the *Join Tables* module you can add population data (from the table on the right) to a points theme containing the names of cities (the table on the left), using the common *City Name* field.

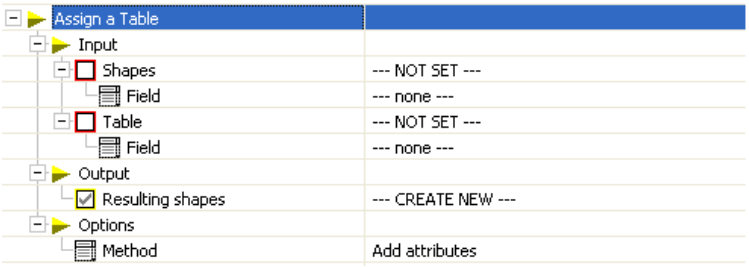
The new attributes table of the points layer will be like this.

City ID	City Name	Population
1	Madrid	5086635
2	Barcelona	3765994
3	Sevilla	1180197
4	Zaragoza	638535
5	Alicante	380357

For a more practical example using real data, open the strassen.shp file that you will find in your demo data folder. Also, open the table in the strassen_typen.dbf file. If you open the attributes table of the shapefile and the table in the dbf file, you will notice that they both have a field named *strTypID*. It contains a code used to define the different types of streets

and roads that are found in the shapefile. However, the description of each type does not appear in the attributes table, but only in the strassen_typen table. We can join both tables so when you open the strassen.shp file you don't have to open the other dbf file to see the description of each street type. This description will be in a new field of the attributes table.

Select the *Join Table* menu item.



Select the strassen.shp vector layer in the *Shapes* field and the strassen_typen table in the *Table* field. In the *Field* lists, select *strTypID* in both of them. In this case, the fields used to join have the same name in both the attributes table and the standalone table, but that is not necessary. They can have different names, as long as they contain the same kind of information, of course.

If no coincidence is found for a particular value, the additional fields will be set to zero if they contain numerical values or to *–Not Set–* if they contain strings.

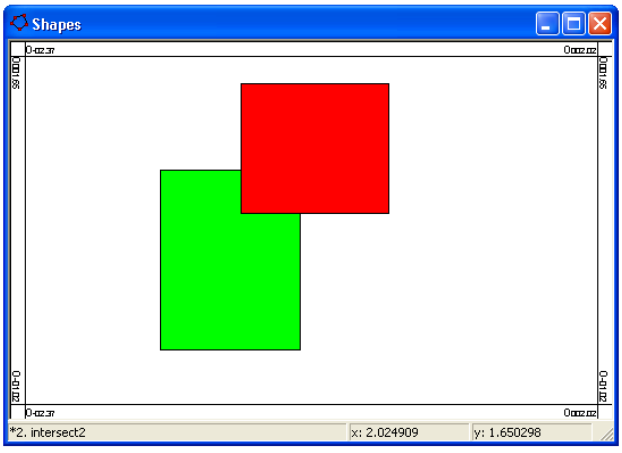
10.6 Intersecting polygon layers

We have already seen how to perform an intersection of raster layers to obtain a new one with new information. We needed some grids and a look–up table, and a couple of little "tricks" as well (remember those prime numbers?).

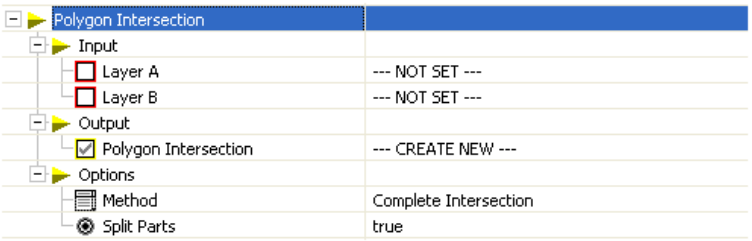
For the Curve Number case, if instead of having grid layers with soil and land cover we have that information in vector format, we can intersect them without having to convert them to raster format, and get a similar result (although, of course, a vector one).

As you might have already guessed, vector intersection can only be performed using polygon layers, not point or line layers.

The polygon intersection module is a rich tool that can be used not only to intersect two layers, but also to perform other different (but closely related) operations. To work with all these functionalities, we will use two very simple polygon layers. In the demo.zip file you can find two files named intersect1.shp and intersect2.shp Open them. Each layer contains just a single polygon. Both polygons overlap, as it is shown in the following picture.



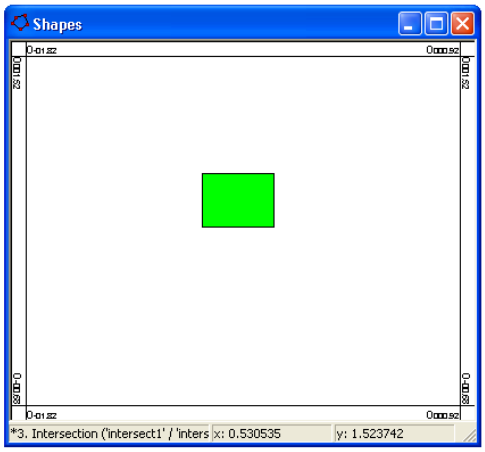
To start the polygon intersection module, select the *Polygon Intersection* menu item.



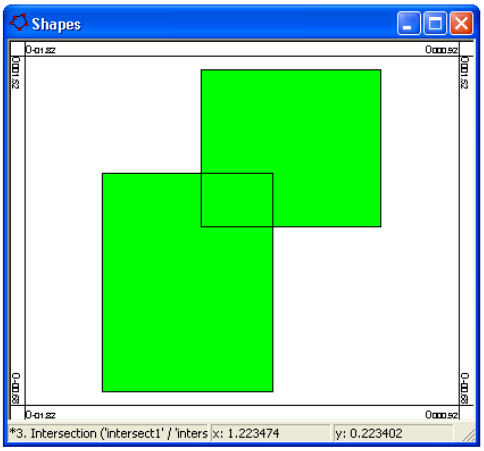
Two layers must be selected in the *Layer A* and *Layer B* fields, and an output layer must be defined.

After the intersection, polygons might get splitted. If you want to keep disjoint polygons in the same shape (a multipart shape), set the *Split Parts* field to *false*. If not, set it to *true* and each part will be in a separate shape.

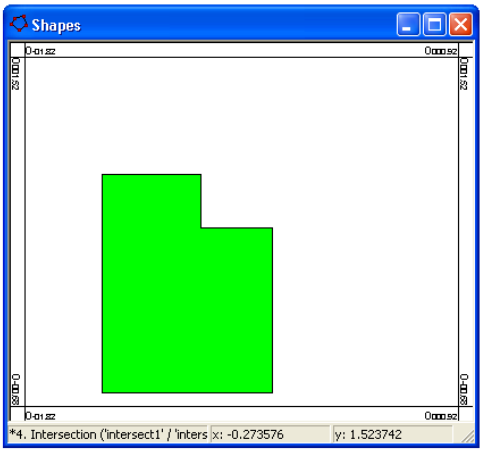
Finally The *Method* field defines the operation to perform. They are all quite self–describing, so instead of explaining what each one does, here is the result of each one of these operations. I'm sure that this is worth far more than any text description.



Intersection



Complete Intersection



Difference

We know now what happens with the polygons, but... what about the attributes tables? Attributes tables of the resulting layers do not contain data from the intersected layers, but just the three following fields:

- **ID**: A code for each polygon created after the intersection.
- **ID_A**: The code of the polygon from layer A from which the new polygon has been created. If no polygon from layer A overlaps with the new polygon, a zero value is assigned.
- **ID_B**: The code of the polygon from layer B from which the new polygon has been created. If no polygon from layer A overlaps with the new polygon, a zero value is assigned.

If using the *Difference* method, the *ID_B* field will not be found in the table.

For example, here is one of the resulting attributes table. Try to guess to which operation it belongs.

Shapes: Intersection ('intersect1' / 'intersect2')

	ID	ID_A	ID_B
> 1	1	1	1
2	2	1	0
3	3	0	1

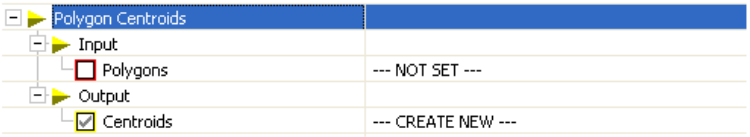
Using the *ID_A* and *ID_B* fields, and a look up table, you can create a new field containing other parameter such as Curve Number or similar. Think about how to do it using some of the ideas introduced in chapter 7.

10.7 Calculating geometrical properties of polygons

Two modules can be used to calculate some geometrical stuff from a polygon layer: *Polygon Centroids* and *Geometrical Properties of Polygons*

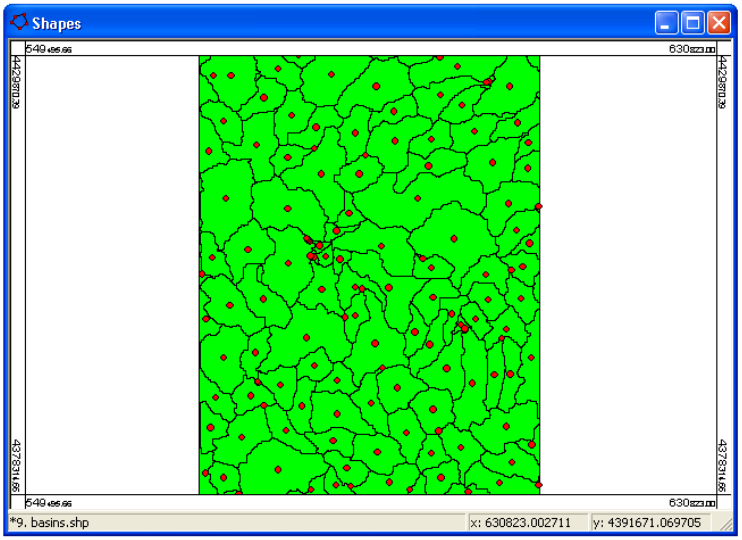
The first one will create a new points layer containing the centroids of all the polygons from a selected layer, while the second one will add two new fields to the attributes table of that layer. Let's see how they work.

Select the *Polygon Centroids* menu item.

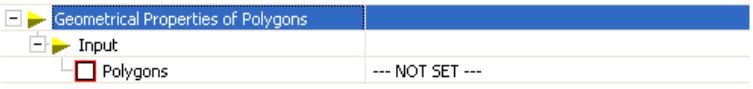


Select the output and input grids, press the *OK* button and you will get a new points layer.

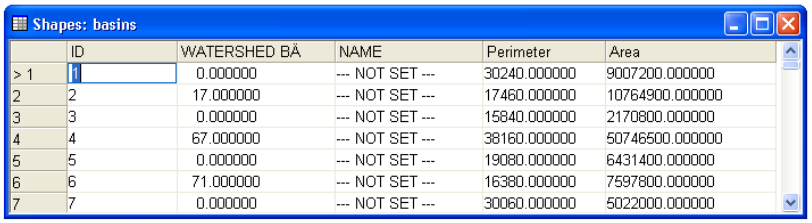
In the following picture you can see the basins vector layer along with its corresponding centroids layer.



The *Geometrical Properties of Polygons* module is even easier to use. No output layer must be selected, since the result will be added to the attributes table of the input one.



Two new fields named *Area* and *Perimeter* will appear on the attributes table. Area and perimeter values are expressed in the same units as vector layer coordinates.

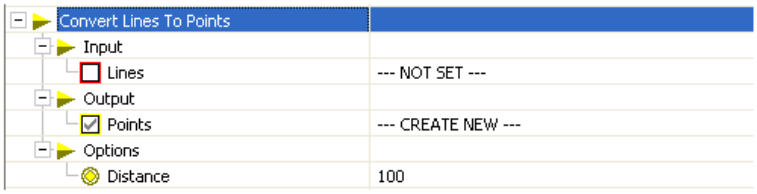


	ID	WATERSHED BÄ	NAME	Perimeter	Area
> 1	1	0.000000	--- NOT SET ---	30240.000000	9007200.000000
2	2	17.000000	--- NOT SET ---	17460.000000	10764900.000000
3	3	0.000000	--- NOT SET ---	15840.000000	2170800.000000
4	4	67.000000	--- NOT SET ---	38160.000000	50746500.000000
5	5	0.000000	--- NOT SET ---	19080.000000	6431400.000000
6	6	71.000000	--- NOT SET ---	16380.000000	7597800.000000
7	7	0.000000	--- NOT SET ---	30060.000000	5022000.000000

10.8 Converting lines to points

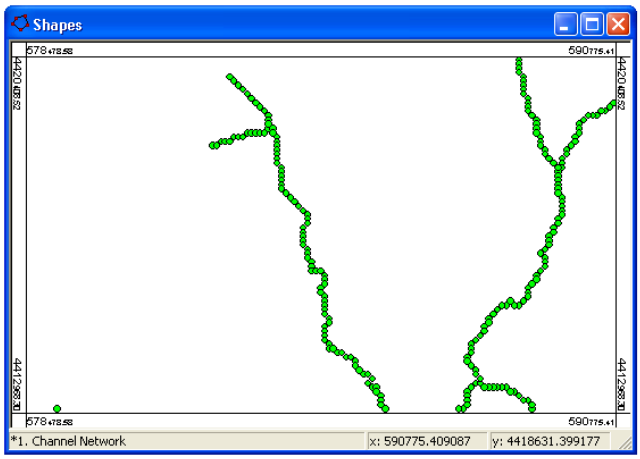
As we will see in the chapter dedicated to interpolation modules, grids can be created from a points layer using several different interpolation techniques. However, height information is rarely found as points, but as contour lines. Those lines cannot be used as input layer by the aforementioned modules, and must be converted to points.

The *Convert Lines to Points* module takes all the nodes of a lines layer and creates a new theme with those nodes as point. It also adds additional points between each pair of nodes, separated by a fixed user–defined distance.



The parameters window of the modules is very simple. Just select the input layer (must contains lines) and an output one (which will contain points). Introduce the distance you want to use to separate points in the *Distance* field. This distance must be in the same units as cell size.

Using the channel network vector layer and a distance of 500 meters, you will get something like this.



When using this module to prepare vector data for interpolation, the choice of a distance must depend on the characteristics of the terrain and the contour lines (equidistance...). Also, the number of points (which depends on the chosen distance) affects the performance of interpolation modules. When we reach the *Interpolating Data* chapter you will have a more solid theoretical base to make the right choice.

10.9 Creating point grids

Although rather different to the previous one, the *Create Point Grid* modules can also be a very valuable tool when interpolating data. It creates a point layer where points are regularly distributed forming a grid. The dimensions of the grid and the space between points are the only required settings.

What can this be useful to? I will show you a real case which I came across not long ago.

Spanish Forestry Inventory was developed using a grid with points separated by 2 kilometers. A sample was taken in each point, measuring information about all the trees within a fixed area from that center point. Given all the samples that contain individuals of a particular specie (in this case was *Quercus suber*), how would you create a grid containing density values of that specie?

The most straightforward thing to do is just take those point and interpolate their density values, isn't it? I'm sorry to say that this option is wrong. Consider two points separated 4 km and no other point between them (remember, you only have those points where a particular specie is found). Interpolating the density values, the cell situated between those samples (where a sample with no *Quercus suber* individuals exists, although you don't have its information) will get a value that will be close to the average of the two know densities (for the sake of simplicity I'm neglecting the influence of other points...). Is that a good value? No! That cell should have a zero value!!

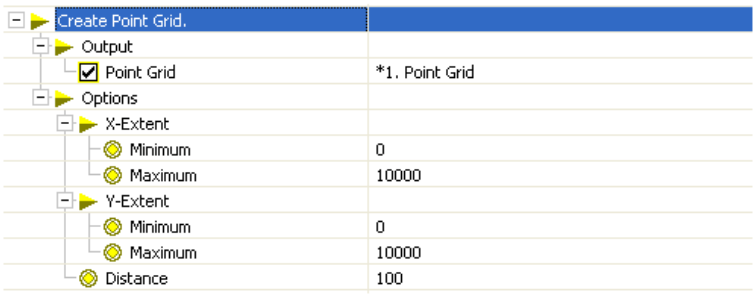
What happens here is that we just have a portion of the original sampling grid, and we must work with the whole grid instead. Otherwise, there is not enough data to correctly create the density grid. Consequently, we must create a *dummy* grid and fill it with zero values to complete the grid. Also, points with known density values must be removed from this dummy grid. Using some basic vector editing functions and the *Merge Layers* module this can be accomplished rather easily.

This lack of information is not only related to the characteristics of the grid, but also to the variable being interpolated. If we were interpolating not the density of an specie, but its average height, there would be no need of filling the grid with zero value points (in fact, that would be wrong). Points were no individuals are found must have a height value close to the average height of the surrounding ones. Right now, no individuals are found there, but if they were, it is quite logical to think that their characteristics would be an average of those found in their neighbors.

The key here is that density is not a continuous value, while average height can be assumed to be continuous. (Think about that. Better yet, look for some references about statistics and learn some more. This kind of problems are one of the most interesting areas of spatial analysis and GIS applications)

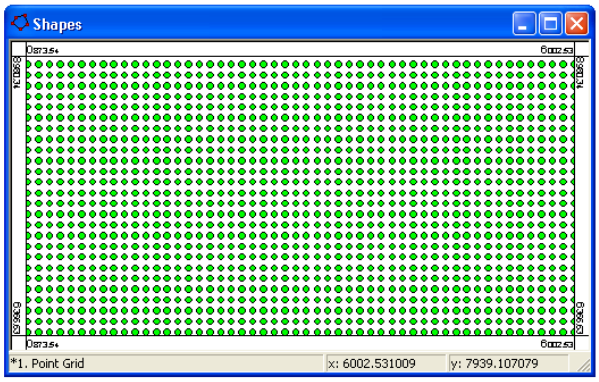
After all that (rather boring) theory, let's see how to create a point grid, whatever the use you are going to give it.

Select the *Create Point Grid* menu item.



Enter the coordinates of the corners of the grid to create, and the distance between points in grid units. Negative values can also be used for the extent parameters.

The new vector layer will look like this.

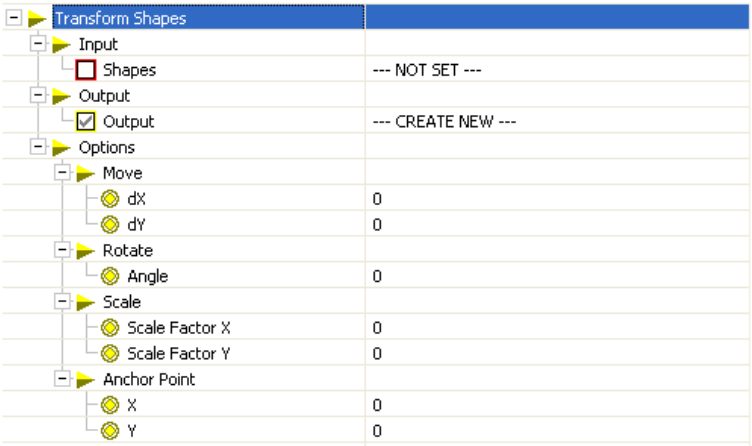


10.10 Moving, rotating and scaling shapes

Shapes in a vector layer can be moved, scaled and rotated using three very handy modules. Although this operations might seem rather imprecise from the cartographer's point of view, if you work abundantly with vector data you will come across many different situations where moving or scaling shapes might be useful to adapt some data or perform some "tricks" that might ease your work.

Some typical uses are rescaling to change the units of measurement (for example, from meters to feet) or changing the origin of a coordinate system.

The *Transform Shapes* module performs these three operation (or just some of them) using just one parameters window.



Leaving the default values will not transform the shapes at all, so if you just want to rotate a layer, leave the parameters under the *Move* and *Scale* nodes unchanged.

If you want to move the shapes in a layer, just enter the x and y offsets in the *dX* and *dY* fields.

To rotate a layer, introduce the angle in degrees and select an anchor point, entering it x and y coordinates. Angles are measured clockwise from north, and negative values can be used as well.

The anchor point is also used to scale the shapes of a layer. You must define the scale factor in both horizontal and vertical direction. For example, to make every shape twice as big, introduce a value of 2 in the *Scale X* and *Scale Y* fields

Chapter 11

Interpolating Data. Geostatistics

11.1 Introduction

Much of the data you are going to work with will not be in raster format. Starting from height information, which, as we have already seen, is frequently represented using contour lines, many variables are stored using vector layers. Since many modules (most of them, in fact) work only with raster data, converting data from one format to another is something that you must learn in order to be able to use all the information that you find.

Learning how to convert data from vector to raster is not only about learning how to use a couple of modules. We have already seen how to create grids from vector layer using the *Shapes Layer to Grid* module, but in this chapter we will study some more complex modules which not just create a raster clone of a vector layer, but can enhance this layer using interpolation algorithms. This requires some extra knowledge about how to adjust this algorithms, so as to avoid getting wrong results or misusing them.

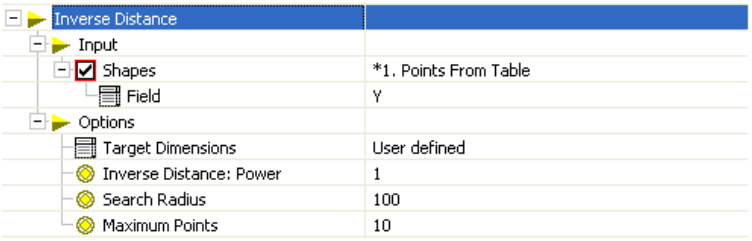
When you interpolate data, you create new data, and the quality of this new data must be taken into account when later using the resulting grids. A couple of basic ideas will help you learn how to create data with enough quality (or sadly realize that your source vector data is not good enough to create a good raster layer).

In this chapter we will work only with points layers, leaving line layers and polygon layers aside. Converting a point layer into a grid with the *Shapes Layer to Grid* module will result in a grid probably containing many no data cells. Using the modules described in this chapter, you will obtain *complete* grids, even if the source vector layer contains just a single point. You will better appreciate this difference when we start working with the interpolation algorithms.

Modules described in this chapter can be found in the Grid_Gridding.mlb library, but also in three other ones named Geostatistics, Geostatistics_Grid and Geostatistics_Kriging. These libraries contain also some modules that do not create grids from vector layers, that will also be described here so as to keep them all together.

11.2 Interpolating data using Inverse Distance Weighting (IDW)

The easiest module to interpolate data can be run selecting the *Gridding/Inverse Distance* menu item.



Select the points layer you want to interpolate and then select in the *Field* list the field where data to be interpolated is stored. Of course, this has to be a numerical field.

Under the *Options* node you will find four fields. One of them is the already known *Target Dimension* one, which needs no further explanation, as do the parameters windows that will appear once you press the *OK* button.

The remaining three fields control the behavior of the interpolation algorithm. For each cell in the output grid, an area of a fixed radius is searched around it for points containing source values. You can set this radius using the *Search Radius* field. Try to select the minimum radius that ensures that a sufficiently large number of points is found around each cell.

Larger radius values will cause module execution to take longer, so you should try to find a correct balance.

You can introduce in the *Maximum Points* field a maximum number of surrounding points to be considered when assigning data to a cell. If this field is set to x, only the values of the closest x points around a cell will be used by the interpolation algorithm.

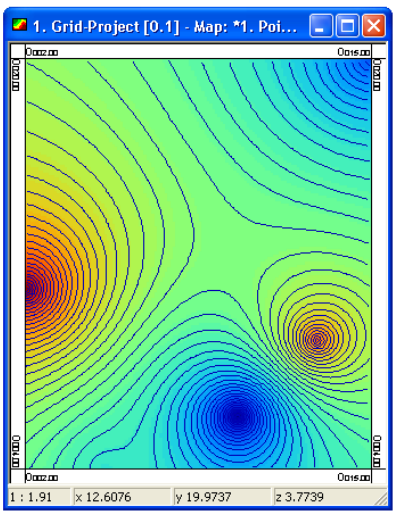
To explain the meaning of the last parameter a bit of math is needed. If N points are found around a cell within the selected search radius, the value assigned to that cell is calculated using the following expression

$$\widehat{z} = \frac{\sum_{i=1}^{n} z_i d_i^k}{\sum_{i=1}^{n} d_i^k} \tag{11.1}$$

where z is the value of the point and d the distance from that point to the cell being interpolated.

The value of n can be introduced in the *Inverse Distance:Power* field. 1 and 2 are typical values that will do for most cases.

Interpolating the points layer created from a coordinates table (go back to 10.3 if you do not remember) yields the following result (a cell size of 0.1 meters has been used, and I have added some contour lines so you can see more clearly the resulting *relief*).



11.3 Creating Thiessen Polygons

The *Inverse Distance* module, as well as other interpolation modules that we will shortly see, always creates a continuous grid. For most variables this is preferred, but discrete grids are a valuable tool too.

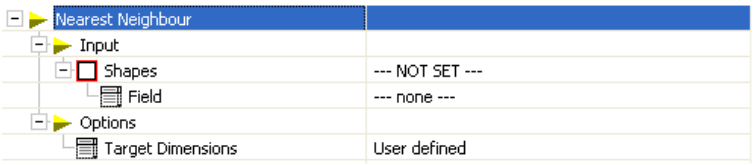
Probably you have already heard about Thiessen Polygons. Given a set of points, each of them has it corresponding Thiessen polygon which represent all those areas that are closer to that point than to the remaining others. In mathematical literature, Thiessen polygons are usually referred as constituting a *Voronoi tessellation*.

When to use Thiessen polygons? Say you have points that represent Metro stations, each one of them with a distinct number (which is kept in an ID field in the attributes table). Calculating Thiessen polygons will tell you which areas have each stations as their closest ones, thus meaning that people from those areas will surely use that station. This way you can see, for example, which one will get more people.

Interpolating the values from those points using IDW will have no sense at all.

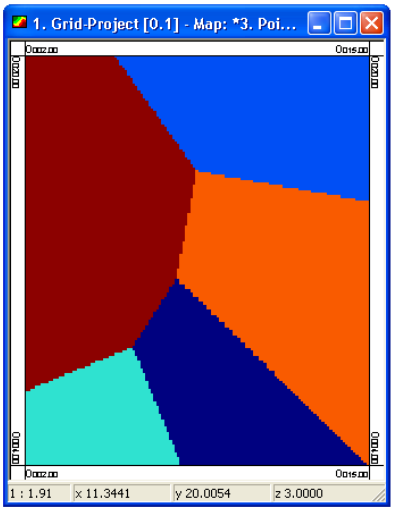
Thiessen polygons are primarily a vector result, but they can also be calculated under a raster scheme. To do it, you just have to use the nearest neighbor interpolation algorithm, so the value of each cell is the one of its closest data point.

To run the corresponding module, select the *Nearest Neighbour* method.



As you see, the parameters window is even simpler, since no additional parameters have to be set for the algorithm itself. I'm sure that you know what to do with the remaining ones.

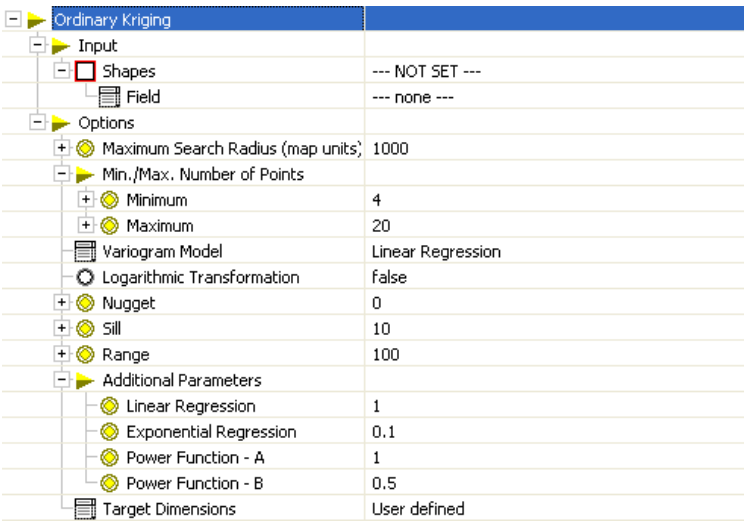
The resulting grid from this method, using the same point layer as in the previous IDW based one, looks as follows.



11.4 Kriging

Kriging is a complex geostatistical technique used to create regular grids from irregularly spaced data points. Unlike IDW, which requires just a few parameters to be adjusted, kriging

is much more difficult to use, as can be seen in the following parameters window, which corresponds to the *Ordinary Kriging* module.



It's not the purpose of this book to introduce the rather complicated ideas that lay under the kriging algorithms (that will take many pages and will surely not be as good as any book you can read on the subject), so only the information regarding module usage will be described here.

SAGA includes four modules to perform kriging interpolation.

- Ordinary Kriging
- Ordinary Kriging (Global)
- Universal Kriging
- Universal Kriging (Global)

The modules containing *(Global)* in their names are just like the ones without it, but a bit simpler, since a maximum search radius is not used and the weighting matrix is generated once globally for all points. Consequently, you will find none of the fields under the *Min./Max.Number of Points* node in their parameters window.

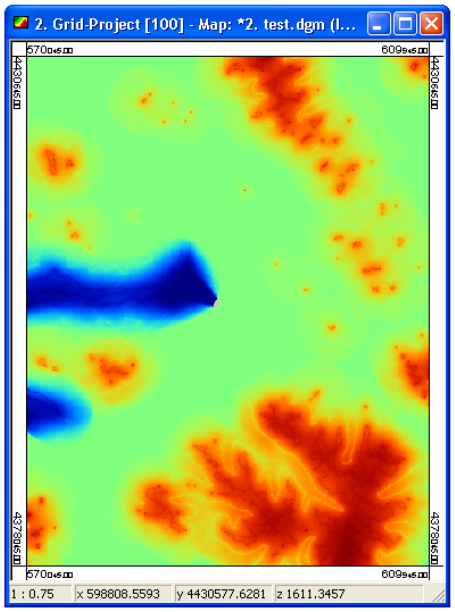
The universal kriging, which should be used if there is a gradual trend in the data, is a bit different. You will not find any of the usual ways of defining grid dimensions. Instead, dimensions are defined using the extension of the grids introduced in the *Grids* field, an input also required for this type of kriging.

In the case of the ordinary kriging, two output grid are always generated: one containing the interpolated values and one containing variance information. When using universal kriging, you can choose which ones of these possible results you want to create.

Now for a little example try this:

- Take the test.dgm grid and create contour lines from it (an equidistance of 500 meters will be OK for practical purposes)
- Convert this lines into points.
- Interpolate this points data using the *Inverse Distance* module.

Using the default values for its parameters (except for the search radius, change it to 3000 or you will get some no data cells), you should obtain the following grid.



It doesn't really look like the original DEM, and it is not difficult to realize that it is not a very accurate terrain representation. Why is that? First, I suggested an equidistance of 500 meters for the sake of speed. Using an equidistance of 20 meters will generate several times more contour lines (and, consequently, more points), thus meaning a rather longer execution time for the interpolation modules.

However, contour lines themselves are rather accurate and contain a lot of points very close one to each other whatever the distance you entered in the *Convert Lines to Points* module. That means that most cells will reach the maximum number of point using points from the same line that have the same height value. That is the main reason for the large green flat area to appear in the middle of the grid. Also, notice how the grid looks better in high areas, since the relief is more mountainous there and contour lines are thus closer.

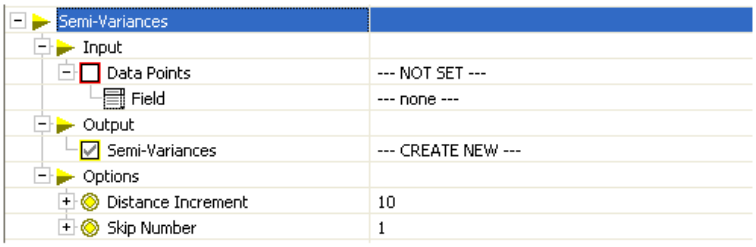
The purpose here is not to create a very accurate grid (we have the test.dgm grid already), but to show you how to get the most out of your points data using the module settings. Therefore, it is now your turn to enter different values and compare the results you get. Try larger search radius or increase the maximum number of points. If you have patience (or a supercomputer), try using contour lines with a higher level of detail

In case you want to practice more (way to go!), do the same using one of the kriging modules instead and check your results against the real DEM.

11.5 Calculating Semivariances

As it was said at the beginning of this chapter, the geostatistical libraries where kriging modules are stored also contain some other modules that, although not aimed at the creation of regular grids from scattered point data, are closely related to some concepts used in the modules already described (specially to the kriging ones), and will be thus explained in this same chapter.

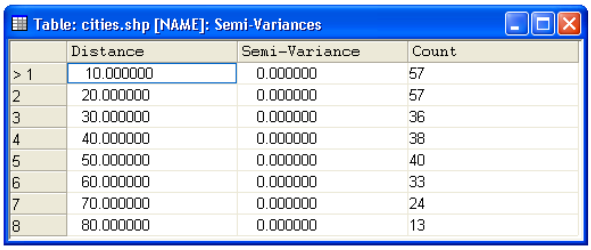
The first of these modules is the *Semi-Variances* one.



As input, you need a points layer. Select it in the *Data Points* field, and then select the field where the data to use can be found.

Adjust the values in the *Distance Increment* and *Skip Number* and press the *OK* button.

The module generates as output a new table named *[Name of Points Layer]: Semi-variances*.

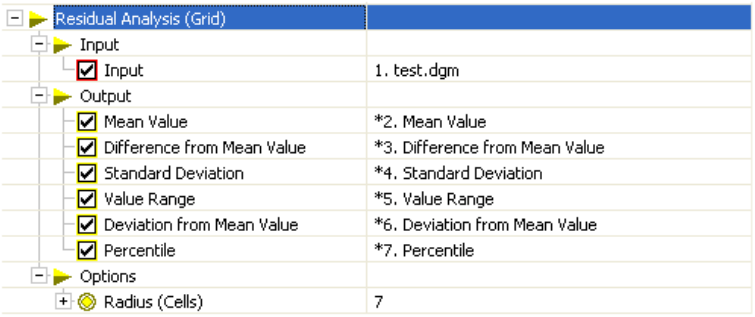


	Distance	Semi-Variance	Count
> 1	10.000000	0.000000	57
2	20.000000	0.000000	57
3	30.000000	0.000000	36
4	40.000000	0.000000	38
5	50.000000	0.000000	40
6	60.000000	0.000000	33
7	70.000000	0.000000	24
8	80.000000	0.000000	13

11.6 Residual analysis for grids

Along with the previous module, which took a vector input and generated a table, three modules can be found that perform their calculations on grids and generate just gridded results. These modules can be found under the *Geostatistics/Grid* menu.

Probably the most useful of them is the *Residual analysis (Grid)* one, which generates a large number of grids containing simple statistical parameters that can be used for many different purposes.



As you can see, most of the fields in the parameters window of this module are related to output grids, input fields being rather limited.

The way all of this output grids are calculated is quite simple: for each cell a fixed number of other cells in its neighborhood are considered, and the values of these cells are used to calculate all of the statistical parameters.

The number of cells to be considered can be adjusted as usual, defining a search radius (in cells) in the *Radius (Cells)* field. This defines what is usually called the analysis window of a cell. Remember that the larger the radius, the more cells that have to be analyzed and the longer it takes for the module to execute.

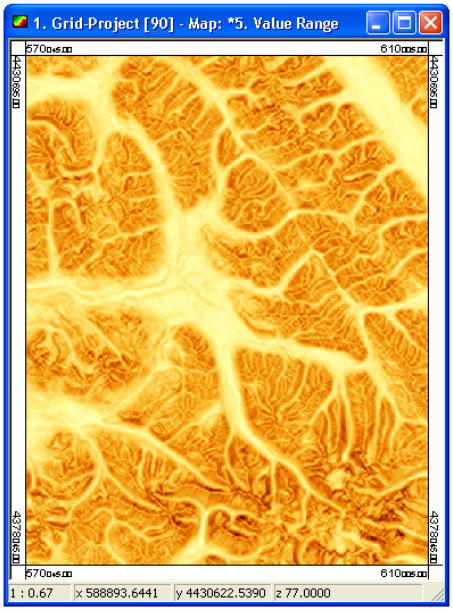
The following grids are calculated. Their corresponding mathematical expression are given when needed. For the sake of space, only some of the resulting grid images are included.

- **Mean value**: The resulting grid will look much as if you passed a smooth filter over the grid. The larger the radius, the more blurry the output image will be.

- **Difference from mean value**: Equals the value of the cell minus the mean value of its surrounding analysis window.

- **Standard deviation**:

- **Range**: Difference between the maximum and the minimum values found in the analysis window.
  When working with a DEM, some of this statistical parameters can be assigned a morphometrical interpretation. For example, this range is obviously related to the slope of each cell. Have a look at its grid and notice how similar it is to the slope grid.



  For a more accurate comparison, try a regression analysis. Use a small radius, as most algorithms for morphometric analysis use a 3×3 analysis window.

- **Deviation from mean value**:

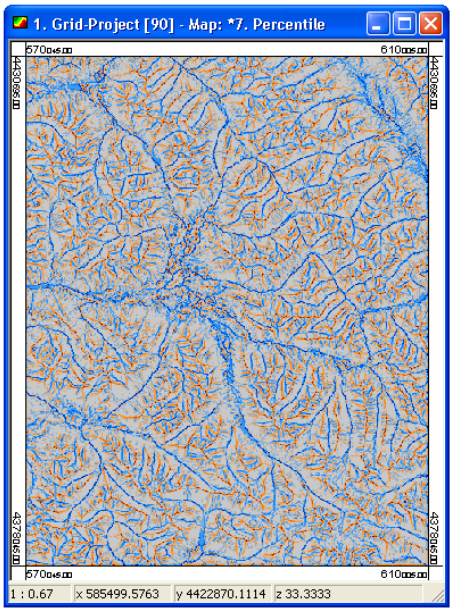
$$D = \frac{z - \overline{z}}{S} \tag{11.2}$$

  Where z is the cell value,
  $\overline{z}$ is the mean value of the analysis window
  and S is the standard deviation.

- **Percentile**:

$$P = \frac{100 N_l}{N - 1} \tag{11.3}$$

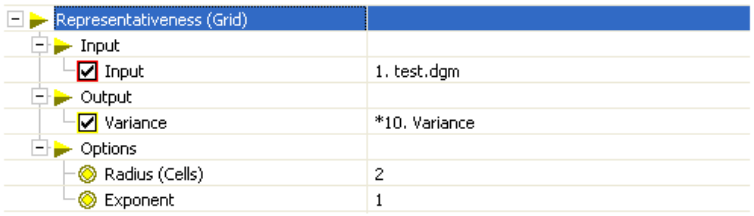
  Where N_l is the number of cells in the analysis window lower than the central one and N is th total number of cells considered.



Try to figure out which morphometrical parameter this grid is related to...

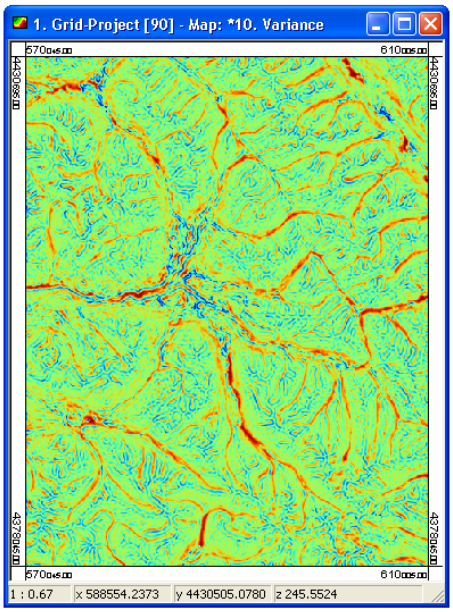
11.7 Representativeness

Unlike the previous module, the *Representativeness (Grid)* one generates a single grid as output which just contain variance values calculated within a given search radius.



This search radius has to be entered in the *Radius(Cells)*, having the same meaning as in the last module.

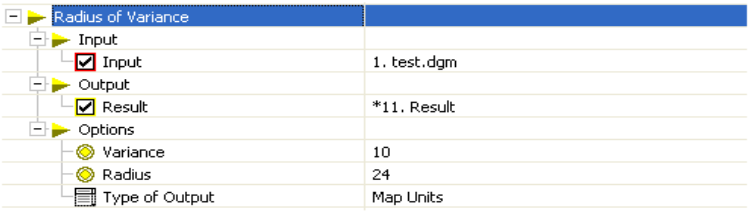
The representativeness grid for a radius of 5 cells looks like this.



Since its results admit several different interpretations, you are not likely to use this module as often as the *Residual Analysis*.

11.8 Radius of variance

Closely related to variance, the radius of variance can be calculated using the homonymous module.

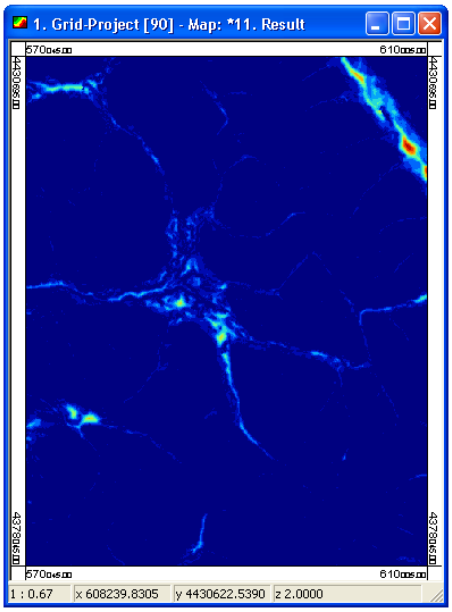


This modules evaluates the variance in the neighborhood of a cell using increasing radius values to define that neighborhood. The minimum radius needed to reach a particular variance value is the radius of variance of that cell.

You must introduce the threshold variance in the *Variance* field. To avoid very large search radius being used, a limit radius value can be introduced in the *Radius* field. If once this limit is reached the variance is still lower than the threshold, the limit radius value is assigned to the cell being analyzed.

Radius of variance values can be expressed in cells or in grid units. You can select between this two options using the *Type of Output* field.

Below this lines you can see a radius of variance grid created using the test.dgm DEM.


1. Grid-Project [90] - Map: *11. Result
1 : 0.67
x 608239.8305
y 4430622.5390
z 2.0000

Chapter 12

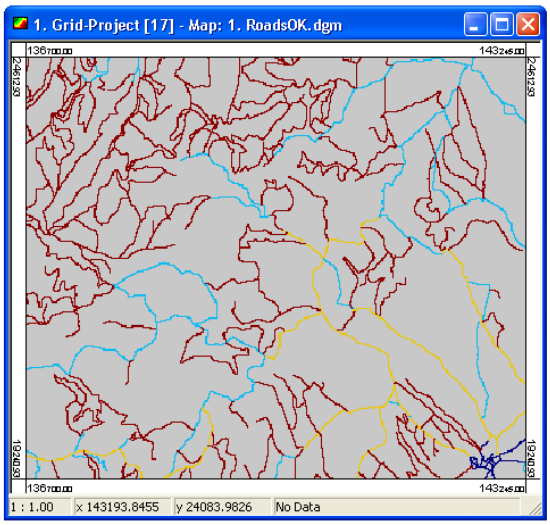
Cost analysis

12.1 Introduction

Another one of the *strong* capabilities of raster GISs is the analysis of costs and the definition of optimal routes through cost surfaces. In this chapter we will study a couple of modules that you can use to perform two different kinds of cost analysis.

Before we get into the modules themselves and start describing their usage, let's see a couple of very simple examples that will help you understand the purpose of this modules and the differences between them.

For the first one, there is a file in the demo.zip file called roads.dgm. It contains roads, and each road cell is assigned a number which represents the average time needed to go through that cell (which is a function of the type of road in it). Points where no roads are found are assigned a no data value.



Sometimes you will have grids containing speeds instead of times. You can rather easily change from one to another using the grid calculator and knowing the cell size value.

Now the question is: from a road cell, which is the fastest route to get to another one?. This fastest path is what we will call the *least cost path*. Cost here is expressed in terms of time: which path takes the least time to get from here to there?

For the second example, picture yourself *inside* our well–known test.dgm grid. At the top of a mountain there is a house and you want to get there but, of course, you want to get there by the easiest path. Unlike in the previous example, cost here is expressed in terms of effort,

you want to find not the fastest path, but the one that takes a lower effort to walk. Also unlike in the case of roads, where the speed was the same whatever the direction in which you crossed the cell, effort here is higher when climbing a steep cell than when going downhill through that same cell.

The first example is based on an *Isotropic* cost surface (i.e. same cost for all directions), while the second one is based on an anisotropic cost surface.

For the two cases depicted above, calculating a least cost path involves the following steps.

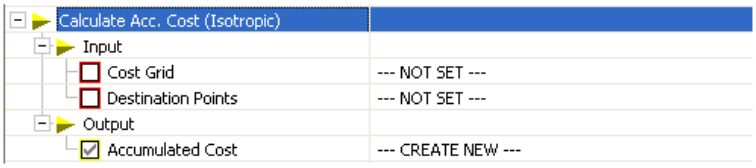
- Preparing a cost surface. If it is an anisotropic one, also a grid with maximum cost direction must be prepared.
- Preparing a grid with destination cells. Those cells must have a non–zero value, while the remaining ones must have a zero or a no data value.
- Creating an accumulated cost surface. It contains not the cost trough each cell, but the overall cost to get from that cell to the closest destination one.
- Selecting an origin cell in the accumulated cost surface. SAGA will trace the least cost path using it, much in the same way as water is routed through a DEM.

Let's see now how to fully develop the two proposed examples using SAGA.

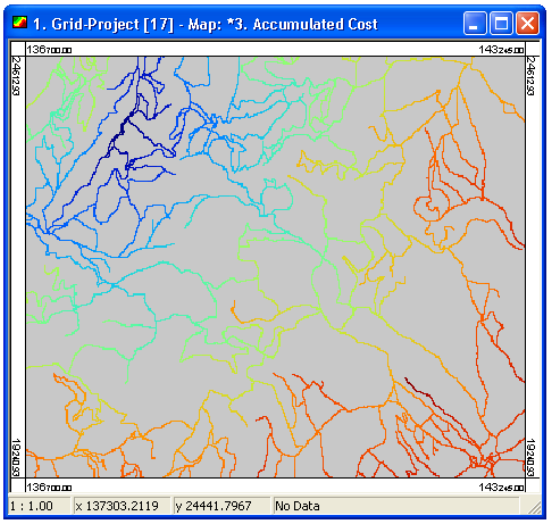
12.2 Creating an accumulated cost surface (isotropic)

We will start with the easiest case. The roads.dgm grid is itself a cost grid, so no further work is required. We can put that grid straight into the accumulated cost module along with a destination cells one. You will find such a grid also in the demo.zip file, named costpoints.dgm. Of course, destination points must fall in cells with a valid cost value. Otherwise, they are ignored.

Once those grids have been loaded, select the *Grid/Cost Analysis/Calculate Acc. Cost (Isotropic)* menu item.



The parameters window doesn't need much explanation. Select the two input grids and press the *OK* button. You will get the following cost grid.



Of course, all of the no data cells remain so in the accumulated cost grid. No data cells are ignored, so the least cost path can never pass through them. That means that, using the roads.dgm grid, only road cells are considered. You can get a similar effect by assigning a very high cost value (and when I say high I mean *very high*, so as to make going through those cells completely unaffordable) to all cells outside the roads. However, this is not recommended, since it can have many drawbacks, as we will see in the case of working with anisotropic cost surfaces. Also, cells with high values are processed (unlike no data cells, which are ignored) thus resulting in a longer execution time.

12.3 Creating an accumulated cost surface (anisotropic)

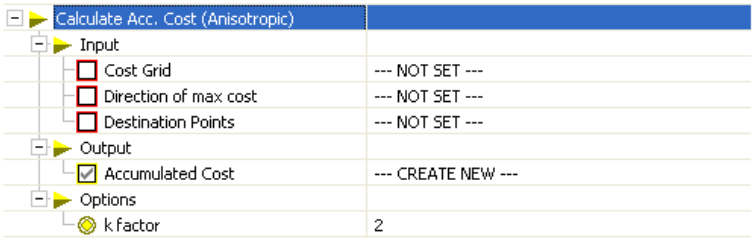
For the second case, things are not so simple. First of all, we need a cost surface, and we do not have it. We have the DEM, but there is no relation between the elevation of a cell and the effort it takes to walk through it. Can you think of a better DEM–based parameter we can use for a cost grid? Yes, the slope is a very good one! The steeper the cell, the more effort it takes to cross it.

For the sake of simplicity, I will use the whole slope grid just as it comes from the *Local Morphometry* module, but you can modify it to get a more real result. For example, you can set a threshold above which cells can be considered impossible to climb. A good idea is to set a no data value range for the slope grid, so all cells above the threshold are considered as no data cells. Don't use here very high values because, since we are working with an anisotropic cost surface, they will not mean a very high cost when cells are crossed in the direction opposite to the maximum cost one. You cannot climb a cell with a very high slope, but also you will not be able to descend across it, since it might be dangerous. So, to completely discard this kind of cells, use just the old good no data values.

Once we have a cost surface, we need a maximum cost directions grid. Guess what we can use? You are right again, the aspect grid!!. Here we cannot (and must not) modify this grid, so we can put it into the cost module just as it comes.

A little adjustment is needed, however, before using the aspect and cost grids. Values in both of them are expressed in radians (remember all the stuff about the Z factor?), and the cost modules needs them in degrees. Use the grid calculator to convert them.

Now select the *Grid/Cost Analysis/Calculate Acc. Cost (Anisotropic)* menu item



The three input grids need no further explanation. You already have them and just have to select them, each one in its corresponding field.

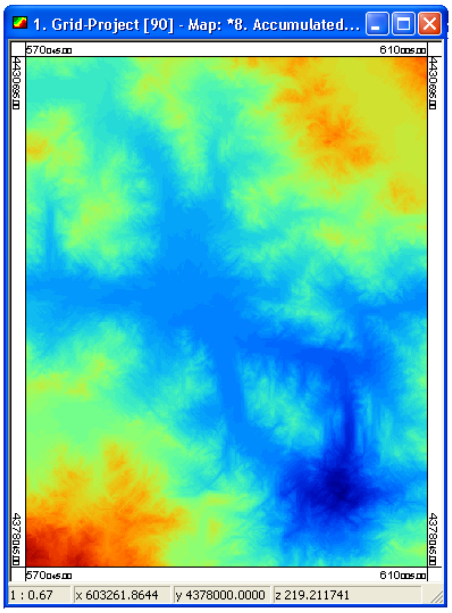
The cost through each cell is calculated using the following equation:

$$F_e = F_s^{\cos^k(\alpha)} \tag{12.1}$$

Where F_e is the effective friction,
F_s is the stated friction (the value in the cost grid),
α is the difference between the maximum cost direction and the direction in which the cell is crossed, k is an additional factor that regulates how cost changes as both angles differ.

You can adjust the k factor using the *k factor* field. The default value is a good choice for most cases.

When done, press the *OK* button and you will get the following grid (this module is rather time consuming, so be patient...)



Quite logically, its global appearance resembles the original elevation grid (think about that...)

12.4 Further preparation of cost surfaces

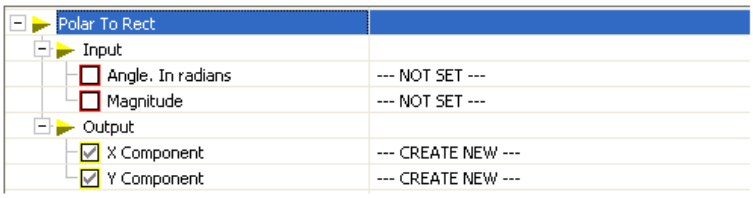
Sometimes, creating an anisotropic cost grid is not as easy as simply calculating slope and aspect grids from a DEM. Some situations require several cost surfaces to be considered, and they have to be all put into one single grid before they can be used as input to the anisotropic cost module.

For example, you might consider the effect of the wind in the above case, so the cost of each cell is also a function of how the wind blows in that cell.

The most tricky part about this is how to add all those individual cost surfaces (each one of them with its accompanying maximum cost direction grid, of course). A very usual way to do it is to convert each pair of values (cost and direction) to a new pair of x and y values reflecting the cost in the directions of the X and Y axis. This way, all X grids can be added to create a global X component grid, and the same with the Y grids. Those two grids can be finally used to create cost and direction grids for the whole set of different cost factors.

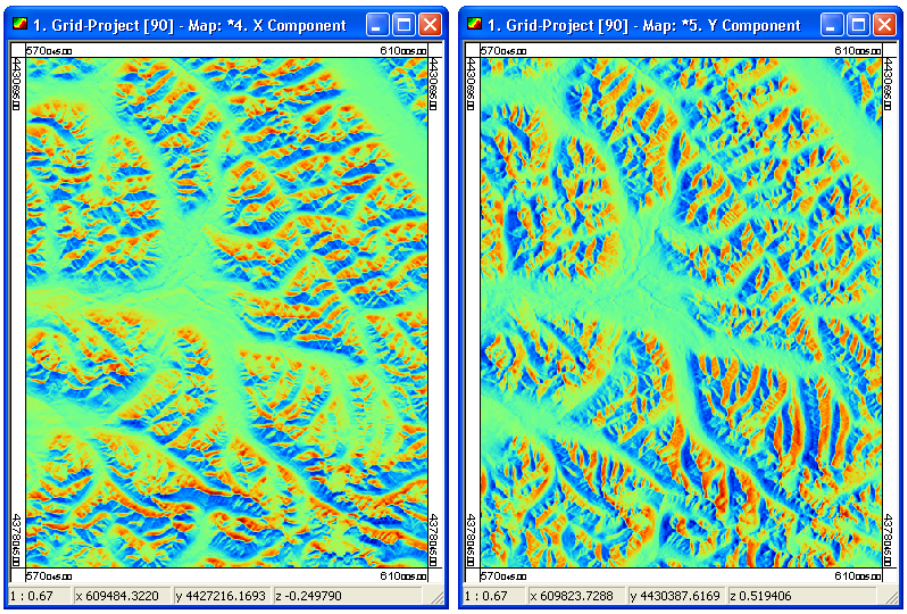
This operations can be performed in SAGA using two modules: *Polar to Rect* and *Rect To Polar*

Select the *Polar to Rect* menu item.

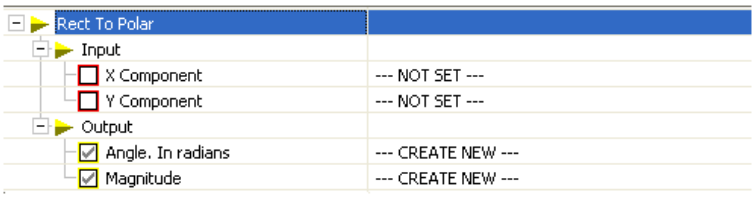


Using this module is pretty simple. Just select the input grids and it will generate the desired X component and Y component ones. The angle grid has to be in radians this time, not in degrees.

Using the slope and aspect grid, you will get the following grids:



The *Rect To Polar* module is almost identical.



I guess you will know how to use it.

12.5 Calculating a least cost path

Calculating a least cost path does not involve using any additional module at all.

You can calculate them using the profile function that is incorporated in the SAGA GUI, and selecting the *Trace Flow* option. As it was said in the *Working with grids* chapter, you must set the *Consider Distance* field to false-

Click on any point of the accumulated cost grid and SAGA will trace the optimal route from that point to the closest (in term of cost, not in terms of euclidean distance) destination point.

Chapter 13

Projections. Georeference

13.1 Introduction

All the information used within SAGA (except for, perhaps, tabular data) must be georeferenced. Without georefence (that is, if the position and dimension of a layer — whether raster or vector — is unknown), many basic tasks such as measuring a distance or combining several layer cannot be accomplished.

SAGA features some modules that will help you to either georefence un–referenced data or change the projection of already georeferenced layers. This is particularly useful when working with several different layers that span a large territory, since (if using the UTM projection) they might have been georeferenced considering different UTM zones. In this case, you can convert all georeferences to a single UTM zone, or (which is less common but far better), using geodetic coordinates instead of projected ones.

These projection modules can be used with raster and vector data indistinctly.

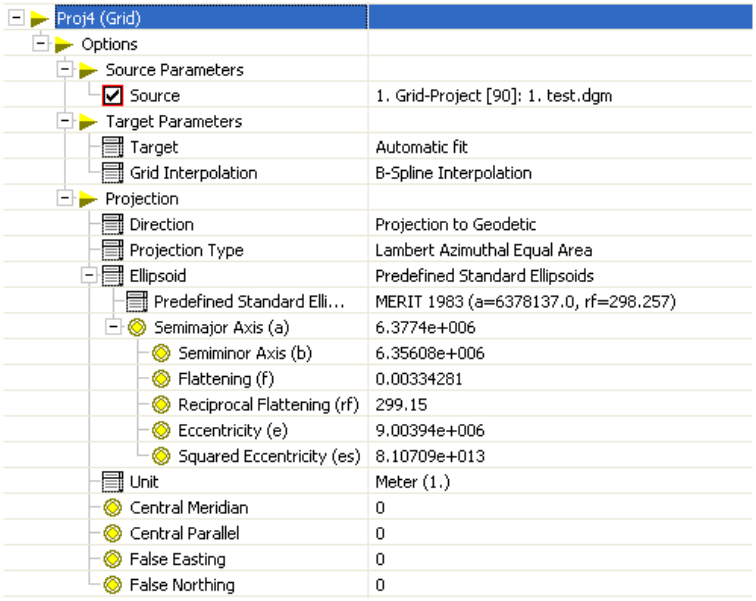
13.2 The Proj4 modules

Wheter you want to change the georeference of a raster or a vector layer, you will find a corresponding Proj4 module. The Proj4 Cartographic Projections library was developed by Gerald Evenden (USGS) and released under a free license (more precisely, a MIT license), and it is used in several relevant GIS projects, SAGA being among them. To find more information regarding this library and its underlying theory, you can go to http://www.remotesensing.org/proj/.

Both the raster and the vector Proj 4 modules share a common usage with just very subtle differences, so only a global description of them will be given, to avoid redundancy.

Make sure that you have a raster layer loaded and go to *Grid/Projection/Proj4(Grid)*. (The vector module can be found in *Shapes/Projection/Proj4(Shapes)*).

You will see the following parameters window:

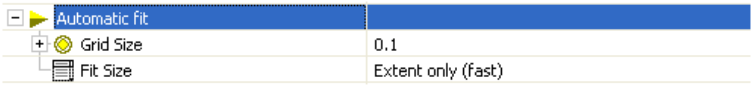


Although it seems rather complex, it is in fact quite easy to use this module. The only thing you can do is to transform data from geodetic to projected coordinates or viceversa, which you must select in the *Direction* field. What if you want to transform a layer from a projection system to a different one? Simply transform it to geodetic coordinates and then from there to the new projection system.

As expected, only one input is needed and only one output is generated. In the case of working with a raster layer, you can choose the type of output in the *Target* field. Along with some options that you will recognize from precedent chapters, you will find two new ones:

- **Automatic Fit**: This option will adjust the extension of the new grid to the smallest area enclosing the transformed data. Have in mind that the transformation might cause no data cells to appear, since it performs operations that might *rotate* the grid.

  If you choose this method, you will see the following parameters window.



  Apart from a cell size for the output grid, select one of the two available methods in the *Fit Size* field. The first is fastest than the second.

- **Shapes**: Instead of a new grid, a new points layer is created, with one point for each transformed cell.

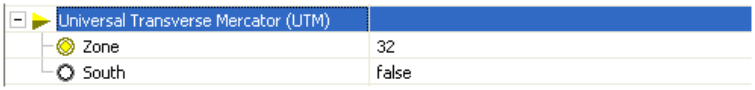
A little warning: when transforming from projected coordinates to geodetic ones and using the *User Defined* or *Automatic Fit* methods you will be prompted for a cell size. Have in mind that cell size is not in meters, but in degrees, so a cell size of 1 will not yield a very precise grid, but a very small one, maybe with just one single cell.

Since grid extent will change from the original grid to the transformed one, an interpolation method is used, which you must select from the list in the *Grid Interpolation*.

If you are transforming a shapes layer it is even easier, and you will find just an output field with the usual options.

The remaining fields are pretty self–explanatory, and you will surely understand how to use them. If you select the *Predefined Standard Ellipsoids* option in the *Ellipsoid* field, you must choose which ellipsoid to use from the list that can be found in the *Predefined Standard Ellipsoids* field. Otherwise, you must define the ellipsoid parameters using the corresponding fields (which depend on the type of ellipsoid you select).

Some projections require additional parameters to be entered. For example, UTM projection needs information about which zone the layer is located in.

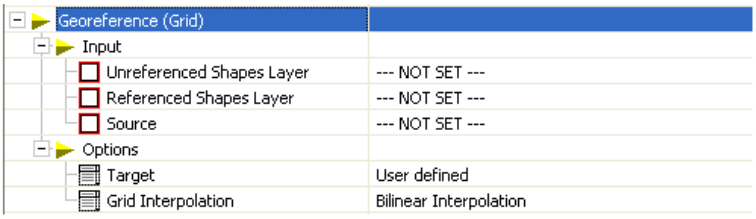


Enter the number of the zone in the *zone* field and set the *South* field to true if the layer is located in the southern hemisphere or to false if it is located in the northern one.

13.3 Georeferencing a grid

To georeference a grid you will need two modules, one for performing the georeferencing procedure itself and another one to prepare the information required by this module.

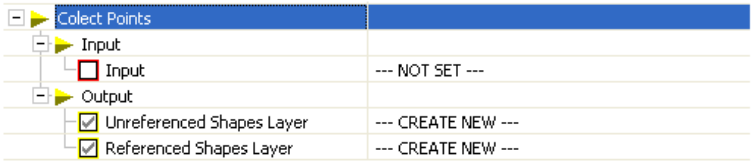
If you run the *Georeference(Grid)* module, you will see the parameters window shown next.



First of all, select in the *Source* field the grid you want to georeference. Also, you need two points layers: one containing point in grid coordinates and the other one containing those same points in "real" coordinates. Select them in the *Unreferenced Shapes* and *Referenced Shapes* fields respectively.

The only two parameters that you can use to adjust how the new georefereced grid is created are the interpolation method and the extension of the target grid. As in the previous module, the *Shapes* and *Automatic Fit* options are available.

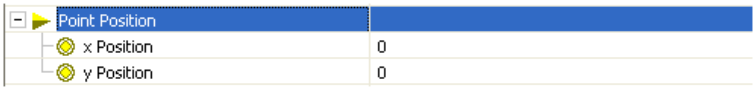
The tricky part here is not how to use the module (which, as we have seen, is pretty straightforward), but how to create those points layers. The simplest way to do that is using the *Collect Points* module.



You can see that the output of this module is a pair of points layers, exactly what is needed for the *Georeference(Grid)* one. As input, you must enter a grid, particularly the one you want to georeference. Do it in the *Input* field.

This is an interactive button, so when you press the *OK* button nothing will happen. Now you can start selecting points on the input grid whose coordinates are known, and for each one of them, a new point shape will be added to each one of the output layers.

When you click on the grid, you will be prompted for the coordinates of that point (the real ones).



Repeat this at least three times, for the georeferencing module will not execute with less than that. Once you have finished collecting points, unload the module clicking again in its menu item.

Chapter 14

More Grid Analysis

14.1 Introduction

Apart from the already described ones, SAGA includes many other modules that, although being rather important, do not constitute by themselves well–defined groups like the ones before. Instead of dedicating a chapter to each one or two of them, I have grouped this modules under this chapter and entitled it *More Grid Analysis*, hoping that you can find in here what you were looking for in case you did not find it in other parts of this book.

I have chosen this title, since almost every one of them takes some kind of grid as input and most of them generate just other grids as output.

I assume that you already have a good command of SAGA and its modular architecture, so the explanations and descriptions in this chapter might be a bit less detailed than in previous ones, inviting you to explore some modules by yourself. However, the meaning of the outputs that each module generates will always be explained.

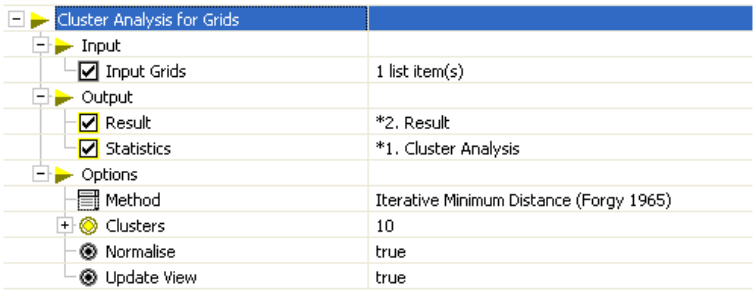
14.2 Discretising grids

We have already seen the difference between a grid containing a continuous variable such as a DEM, and a grid containing a discrete one such as a Curve Number grid. Discrete grids are also useful when used with continuous variables, classifying the variable into classes, since this qualitative grids can be combined with other non–continuous grids to get result that would be more difficult to obtain using just the continuous one.

You can quite easily convert a raster layer such as a DEM into a grid containing classes using the *Change Grid Values* module, which we saw some chapters ago. However, there are better ways to do this than simply reclassifying the values, which can also be used for several grids at once, not just for a single one. We will see in this section three modules that deal with discrete grids, all of which can be found under the *Grid/Discretisation* menu.

14.2.1 Clustering a grid

Probably the most useful one of these three modules is the *Cluster Analysis for Grids*



Using this module, you can convert a continuous grid (or a bunch of them) in a classified one, using algorithms that do not only take into account the value of the cell, but also the values of its surrounding ones and its position within them.

You can select several grids using the *Input Grids* field and selecting them in the usual multiple selection window.

As output, the module creates a new grid with classified information, but also a table with some very interesting statistical values.

Choose a method from the *Method* list. Each one creates a different grid, so try them and select the one you consider better for your purposes.

One of the key parameters of the module is the number of different classes that you want to create. Introduce it in the *Clusters* field. The larger the number of clusters, the more times the module has to iterate, thus resulting in a longer execution time.

Input grids can be normalized by standard deviation before clustering. Set to true the *Normalize* field if you want grids to be normalized.

The clustering process involves iterating and creating several grids before reaching the final one. If you want to see this intermediate grids as they are created, set the *Update View* to true. Module execution will take a longer time, but its quite nice to see how grids evolve (it has a sort of psychotropic LSD–esque feel to it), so try it at least once to see what I am talking about.

For an example, we will not use the test.dgm grid here, but the one that we created in the interpolation chapter using the inverse distance method. Using the Hill–climbing method and introducing a value of 10 in the *Clusters* field, you should get a grid like this:

If you go to the table project window yo will find a table named *Cluster Analysis*. Open it.

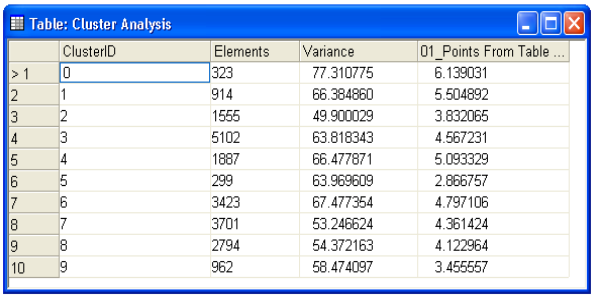


Table: Cluster Analysis

	ClusterID	Elements	Variance	01_Points From Table ...
> 1	0	323	77.310775	6.139031
2	1	914	66.384860	5.504892
3	2	1555	49.900029	3.832065
4	3	5102	63.818343	4.567231
5	4	1887	66.477871	5.093329
6	5	299	63.969609	2.866757
7	6	3423	67.477354	4.797106
8	7	3701	53.246624	4.361424
9	8	2794	54.372163	4.122964
10	9	962	58.474097	3.455557

The table contains information for each one of the created clusters. Each one has an ID (the value in the clustered grid), and you can see the variance and the number of cells in it. Apart from this columns, you will find a new one for each grid that was used as input. In this case, since we just used one grid, there is only one additional column, named *01_Points from Table (Inverse Distance)*. The value in this column indicates the mean value of this grid in the cluster.

If you order the record of the table according to their mean values for the input grid, you can see that a higher ID does not imply a higher mean value. Take care when interpreting the clustered grid, and always go to its assocated table to check the values of each cluster.

14.2.2 Skeletonizing classes in a grid

Skeletonization is the process of *taking away* pixels from a feature (defined by a block of pixels) until that feature is as thin as possible and still recognizable. Skeletonization is widely used in image analysis, but it also has great interest from the GIS point of view.

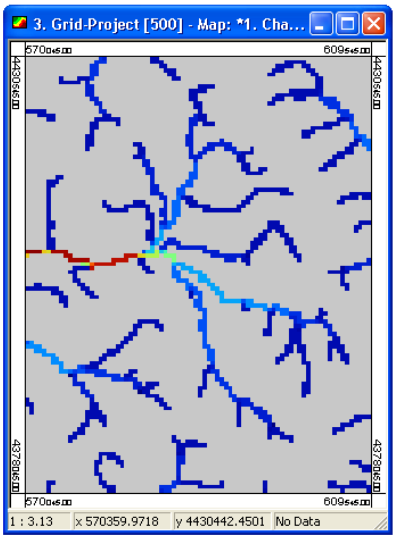
You can use skeletonization to *thin* roads in a grid in case they are more than one pixel wide, leaving only the core information about them. If you later want to vectorize this roads, the narrower they are, the better results you will get.

Skeletonization can be performed in SAGA using the *Grid Skeletonization* module.

This module takes a single (discrete) grid as input and generates a new one containing skeletonized features. Optionally, this features can be output in vector format, a new shapes layer being thus created.

The main options that can be found in the parameters window are the method to use (as in the case of clustering, try them to see which one gives a better result for your particular case), the initiation criterion and its threshold, and a convergence factor.

Let's work out an example so you can see the effect that those parameters have on the resulting skeletonized grid. To prepare an input grid, take the grid containing the channel network that we created on the terrain analysis chapter and resample it to a cell size of 500 meters. Use the *Mean Value* interpolation method to get the following grid.

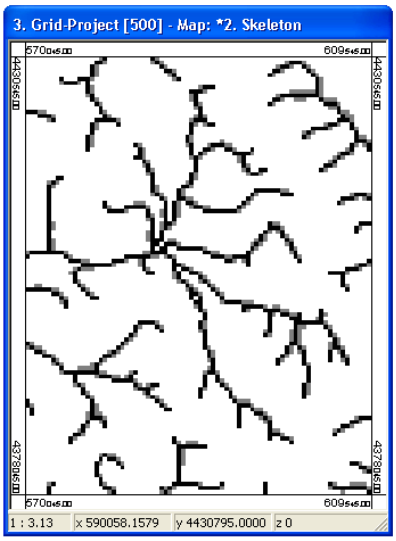


Notice how the resampling has caused some channels to be several cells wide, specially at their higher segments. We will try to get a new grid with narrower channels.

Select this grid as input in the skeletonization module and introduce the following parameters.

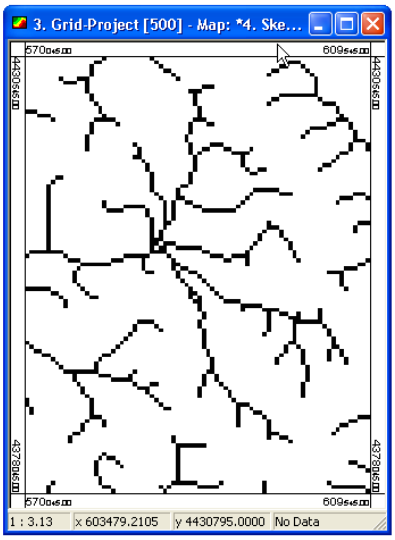
- **Method**: Standard
- **Initialisation**: Greater than
- **Threshold**: 0.5
- **Convergence**: 3

You will get this new grid.



In it, cells outside channels get a value of 0, channel cells get a value of 2, and some cells close to channel cells get a value of 1. You can take away this last cells by changing their value to 1 or, better yet, setting a no data value range from 0 to 1. This way you will get a grid containing the desired thinned channels, defined by cell whose value equals 2.

Below you can see how the final grid looks like.



Let me leave you a little homework: try to get a thinned channels grid but, instead of assigning the same value (=2) to all channel cells, assign to each cell its channel order. Easy, don't you think?

14.3 Some image analysis modules

The number of SAGA modules dedicated to image analysis is quite limited, although the own structure of SAGA makes it quite easy to implement them much in the same way as all the raster modules we have already seen. Some of these raster modules such as the clustering one can be used as a basic tool to perform unsupervised image classification, but that is not their main aim and better tools could be implemented in separate modules.

In this section we will have a look at two very easy modules that allow the extraction of simple but very valuable results.

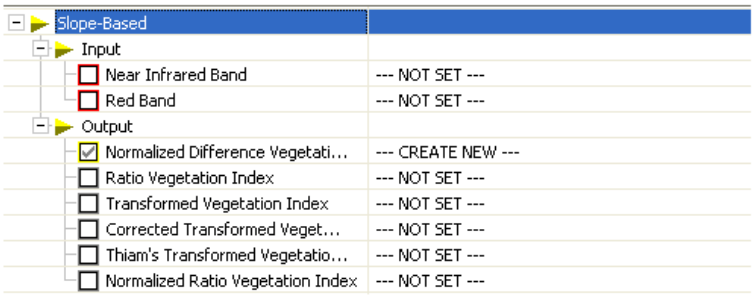
14.3.1 Vegetation Indices

Vegetation Indices (VIs) can be used to estimate vegetation cover or even the vigor of vegetation where it exists. For example, they can be used to derive Leaf Area Index (LAI) by regressing the vegetation index against ground measurements of some plant parameter.

All of the VIs that can be calculated with SAGA just require two input raster layers (two images): one containing infrared band values and another one containing red band values. Both of them can be obtained from a variety of multispectral sensors.

VIs can be divided in to main groups: distance–based and slope–based. Each one of these groups has different requirements, so they are divided in two modules.

The easiest to use is the *Vegetation Indices/Distance–based*. Run it and you will see the following parameters window.



You just have to introduce the two already mentioned images and select which of the grids found under the *Output* node you want to obtain as output.

These are the available indices, each one with its corresponding equation:

- **Normalized Difference Vegetation Index:**

$$\frac{\mathrm{NIR} - \mathrm{Red}}{\mathrm{NIR} + \mathrm{Red}} \tag{14.1}$$

  Where NIR is the near infrared band value,
  and Red is the red band value.

- **Ratio Vegetation Index**

$$\frac{\mathrm{NIR}}{\mathrm{Red}} \tag{14.2}$$

- **Transformed Vegetation Index**

$$\sqrt{\mathrm{NDVI} + 0.5} \tag{14.3}$$

- **Corrected Transformed Vegetation Index**

$$\frac{\mathrm{NDVI} + 0.5}{|\mathrm{NDVI} + 0.5| \cdot \sqrt{|\mathrm{NDVI} + 0.5|}} \tag{14.4}$$

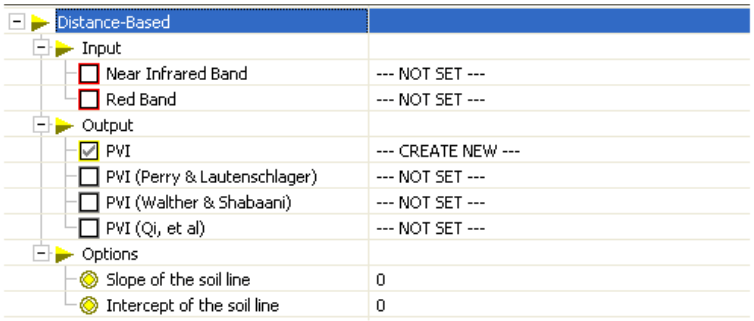
- **Thiam's Transformed Vegetation Index**

$$\sqrt{|\mathrm{NDVI} + 0.5|} \tag{14.5}$$

- **Normalized Ratio Vegetation Index**

$$\frac{\frac{\mathrm{Red}}{\mathrm{NIR}} - 1}{\frac{\mathrm{Red}}{\mathrm{NIR}} + 1} \tag{14.6}$$

Distance–based VIs are a bit more complicated and require some previous preparation. Run the corresponding module selecting the *Distance–based* menu item, and let's have a look at its parameters window



The key here is in the *Slope of the soil line* and *Interception of the soil line* fields, which have to be calculated separately and then entered in the parameters window.

Here is how to do it:

- Calculate a slope–based VI, for example NDVI.
- Using it, select the areas that represent bare soil.
- Create a grid containing 1 in bare soil cells and no data value in the remaining ones (use some masking techniques)
- Apply this mask grid to red and infrared grids
- Perform a simple linear regression using those grids. You will get an equation in the form

$$y = ax + b \tag{14.7}$$

  a is the slope of the soil line and b is the intercept.

These are the available VIs in this module

- PVI (Richardson & Wiegand)
- PVI (Perry & Lautenschlager)
- PVI (Walther & Shabaani)
- PVI (Qi, *et al*)

Depending on the VI you want to calculate, the regression must be performed using the red band or the near infrared one as independent variable. According to this, these VIs are divided in two groups:

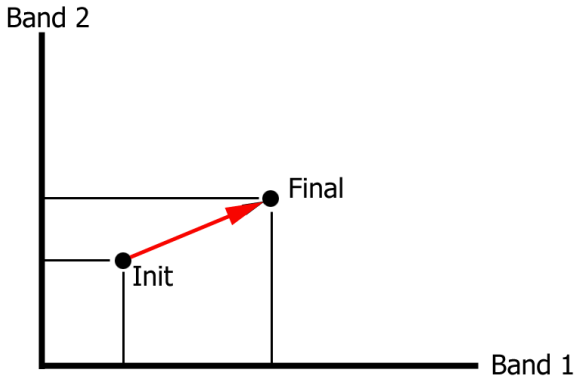
- **Red as independent variable**: PVI (Walther & Shabaani, PVI (Qi, *et al*)
- **Infrared as independent variable**: PVI (Richardson & Wiegand, PVI (Perry & Lautenschlager)

14.3.2 Change Vector Analysis

Unlike vegetation indexes, which produced *static* information, Change Vector Analysis (CVA) shows the variation that occurred in two images (two bands) between two dates. You need, thus, 4 images: two taken the initial date (typically red and near infrared images) and these same two taken the final date.

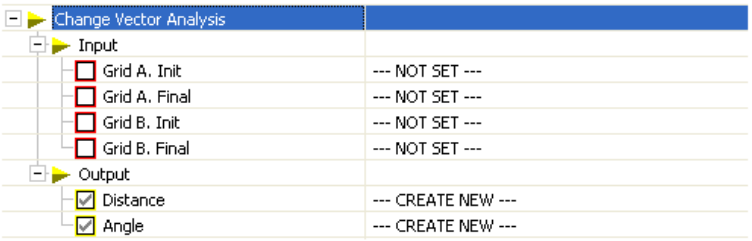
Taking one of this bands (we will call it Band 1) as the X axis and the other one (Band 2) as the Y axis, we obtain four coordinates and two points in the form (x, y). One of this point represents the initial date, while the other represents the final one. The vector from the initial to the final point can be expressed in polar coordinates as (*magnitude*, *angle*).

Have a look at this picture to understand the meaning of all this.



Putting each one of this values in one grid, and calculating this for each cell, we get two new grids (one containing magnitude and one containing angle), with which the variation between the two defined dates can be analyzed.

CVA is performed in SAGA using the *Change Vector Analysis* module.

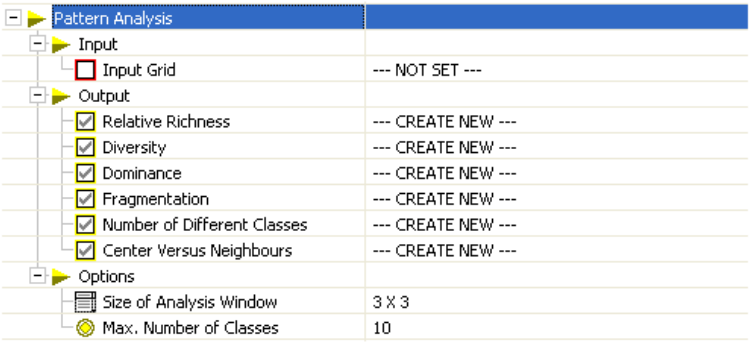


Not much to explain here, I guess...

14.4 Pattern analysis

Pattern analysis is particularly useful for landscape analysis, and it can be performed on discrete grids and also images. There is just one pattern analysis module in SAGA, but it generates a large set of new grids among which you will surely find what you need.

To start this module, select the*Analysis/Pattern Analysis* menu item.



The parameters window is pretty easy to configure. First, introduce the grid (or image) you want to analyze in the *Input Grid* field. Select a size for the analysis window. As usually, larger sizes take longer to be evaluated. Finally, introduce in the*Max. Number of Classes* the maximum number of classes to be considered in the whole grid. Each different value defines itself a class. As we will see, this value is required to calculate some parameters.

And that's all. Press the *OK* button and you will get the following grids:

- **Relative Richness:**

$$R = \frac{N}{N_{\max}} \cdot 100 \tag{14.8}$$

  Where N is the number of classes in the analysis window, $N_{\max}$ the maximum number of classes.

- **Diversity:**

$$H = \sum_{i=1}^{N} P_i \cdot \ln P_i \tag{14.9}$$

  Where N is the number of classes in the analysis window, P_i the proportion of class i in the analysis window.

- **Dominance:**

$$D = H_{\max} - H \tag{14.10}$$

  Where $H_{\max}$ is the maximum diversity ($=\ln N$)

- **Fragmentation:**

$$\frac{N-1}{c-1} \tag{14.11}$$

  Where c is the number of cells in the analysis window.

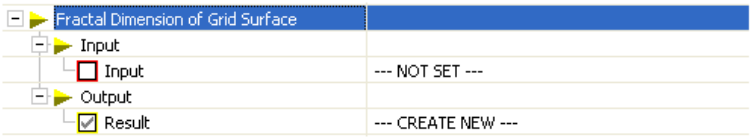
- **Number of different classes:**

- **Center versus neighbour**: Indicates the number of classes in the analysis window different from the center cell.

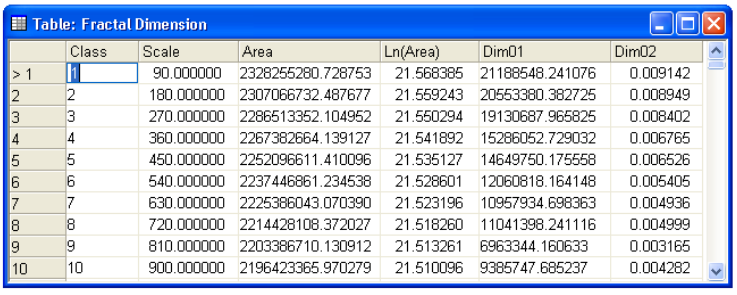
14.5 Fractal dimension of a surface

You can find the *Fractal Dimension of Grid Surface* module under the *Recreation/Fractals* menu. Although fractals can be incredibly fun (I guess that's why the module is located there), they are also very useful, so I believe it is a good idea to include its description here rather than in other chapter along with some truly recreational ones.

The parameters window of this module is very simple and needs no explanation.



The output is a new table like the one shown next.



	Class	Scale	Area	Ln(Area)	Dim01	Dim02
> 1	1	90.000000	2328255280.728753	21.568385	21188548.241076	0.009142
2	2	180.000000	2307066732.487677	21.559243	20553380.382725	0.008949
3	3	270.000000	2286513352.104952	21.550294	19130687.965825	0.008402
4	4	360.000000	2267382664.139127	21.541892	15286052.729032	0.006765
5	5	450.000000	2252096611.410096	21.535127	14649750.175558	0.006526
6	6	540.000000	2237446861.234538	21.528601	12060818.164148	0.005405
7	7	630.000000	2225386043.070390	21.523196	10957934.698363	0.004936
8	8	720.000000	2214428108.372027	21.518260	11041398.241116	0.004999
9	9	810.000000	2203386710.130912	21.513261	6963344.160633	0.003165
10	10	900.000000	2196423365.970279	21.510096	9385747.685237	0.004282

Fractal dimension is calculated using windows of increasing size. That's why the values in the *Scale* column are all multiples of the cell size. For each scale you have its area value and the natural logarithm of it. The two last columns contain differences between each row and the next one for the *Area* and *Ln(Area)* columns.

Chapter 15

Other modules

15.1 Introduction

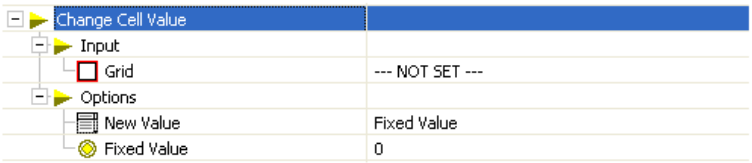
Did you think that you had already seen all of the SAGA modules? Well, almost, but not yet. There are still a few ones left (and probably more yet to come in a near future), which I have gathered in this last chapter about modules.

Most of these are rather simple, and with the knowledge you have already acquired you should be able to figure out how to use them. However, in case you need a little help, here you have a brief description of them.

15.2 Changing the value of a single cell

Remember those interactive modules we saw a few chapters ago? Here we have another one of them. The *Change Cell Value* module can be really useful when you need to change the value of a single cell or just a few ones. It can be used, for example, to create a destinations point grid to be used with the cost analysis modules. Just generate a grid with no data values in its cells and then change the value of some of them to a different value, so they are recognized as destination points by the cost analysis algorithms.

Changing cell values is pretty simple. Run the module selecting the *Change Cell Value* menu item



Select in the *Grid* field the grid you want to modify. That means that the selected cells will be modified in this grid, but you can use any other one from the same project to click on it and select the cells you want to change. For example, you can use the roads grid to ensure that destination point fall on road cells.

After that, select the new value to assign to the selected cells in the*Fixed Value* field. If the *Value Type* field is set to *Fixed Value*, all the cells you select will be assigned this value. If you set this field to *Ask Each Time*, each time you click you will be prompted for a new value. That allows a more flexible use of the module in case you want to assign different values to different cells.

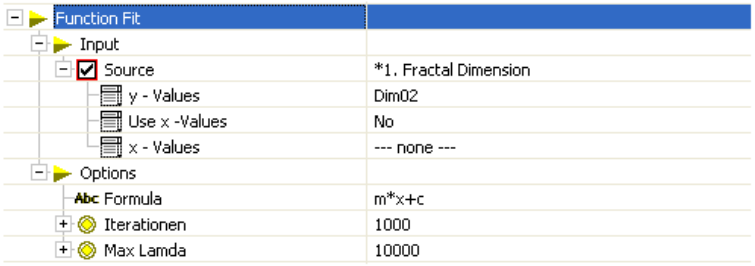
Press the *OK* button to close the window and now you can start clicking on the grid to change (or another from the same project). Each cell where you click will change its value to the one you introduced in the corresponding parameters window.

Once you finish using this module, click again in its menu item to stop it.

15.3 Fitting a function to tabular data

We have seen how to produce some tabular results and how to edit and save them. Saving them is a good idea if you want to perform some further analysis on this data, since SAGA supports many formats also supported by popular applications like spreadsheets or similar, which include a larger set of analysis functions. However, SAGA modules can also be created to perform such operations, so you can get some additional results or interpretations without the need of any other software.

Right now, only one single module for tabular data analysis can be found: *Calculus/Function Fit*.



First of all, select in the *Source* field the table where source data are stored. For this example, we will use the fractal dimension table we created in the last chapter.

Select the column with the y values in the *y–Values* field. If points are equally spaced, as in this case, there is no need to use x values. You can set this in the *Use x–Values* field. In case you want to use them, select the column where they are found in the *x–Values* field.

In the *Formula* field you must introduce the type of the curve to fit. This is a very flexible field, much in the style of the one found in the grid calculator module, but with some additional functionalities.

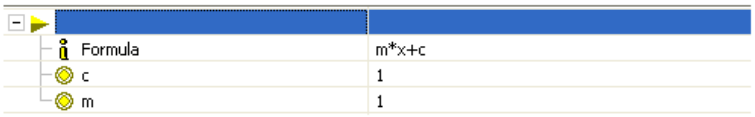
You can use operators such as addition(+) or subtraction(-), along with functions such as cos(), sin() and others that were already explained when we used the grid calculator module. The x value must appear, and then you can add constants using single characters like $a, b, c, d \ldots$

Additionally, you can perform variogram–fitting using the following functions.

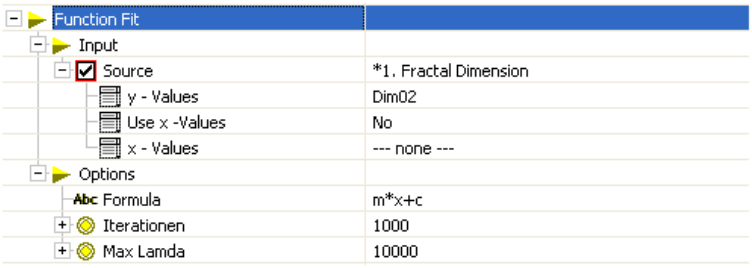
- **SPH(x)**: Spherical model
- **NUG(x,a)**: Nugget
- **LIN(x,a)**: Linear Regression
- **EXP(x,a)**: Exponential regression

Adjust the equation to your needs and then fill the *Iterations* and *Lambda* fields. These two fields control how the iterative process is performed and when to stop iterating, so you should use them to get a finer tuning of the fitting algorithm.

Once all the fields have been set, press the *OK* button. You will see a new parameters window.



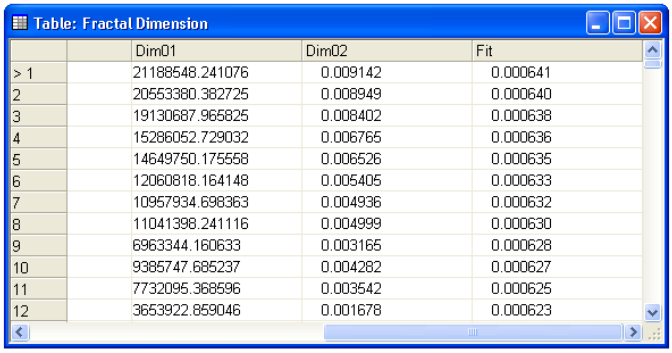
Since the fitting process is based on an iterative scheme, you must supply initial values for all constants you used in the *formula* field



Press the *OK* button again to start module execution.

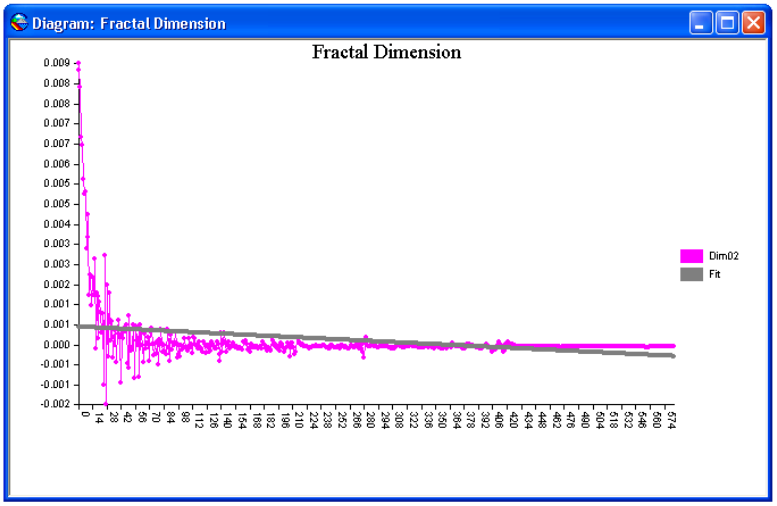
This module does not create any new element, neither table, nor grid, nor vector layer. Instead, it just adds a new column named *Fit* to the source table.

Fitting a linear equation like the one that appears by default in the *Formula* field, and using the *Dim02* column for the y–values, you should get the values shown next.



	Dim01	Dim02	Fit
> 1	21188548.241076	0.009142	0.000641
2	20553380.382725	0.008949	0.000640
3	19130687.965825	0.008402	0.000638
4	15286052.729032	0.006765	0.000636
5	14649750.175558	0.006526	0.000635
6	12060818.164148	0.005405	0.000633
7	10957934.698363	0.004936	0.000632
8	11041398.241116	0.004999	0.000630
9	6963344.160633	0.003165	0.000628
10	9385747.685237	0.004282	0.000627
11	7732095.368596	0.003542	0.000625
12	3653922.859046	0.001678	0.000623

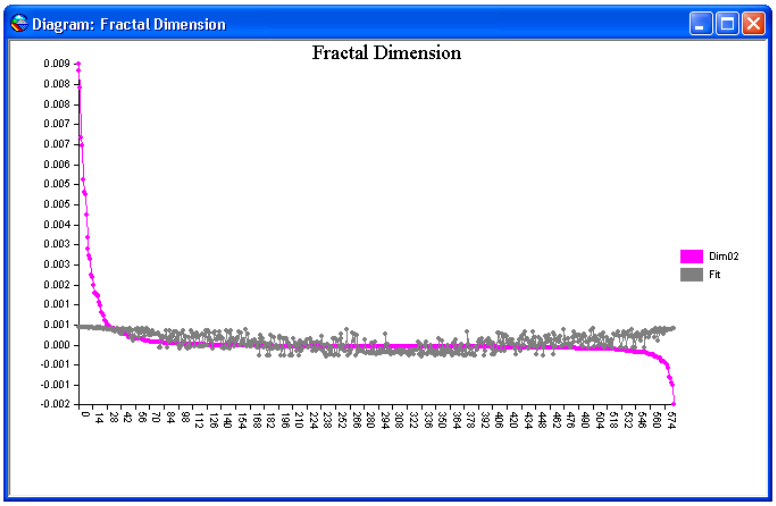
It's a good idea to represent the original values and the fitted ones together, so as to have a better idea of the goodnes of fit. Do it using the *Diagram* function that was explained in the *Working with tables* chapter.



Here's a little trick about tables and diagrams. Whenever you create a diagram from a table, it is dynamically linked to it. That means that if you change the values in the table,

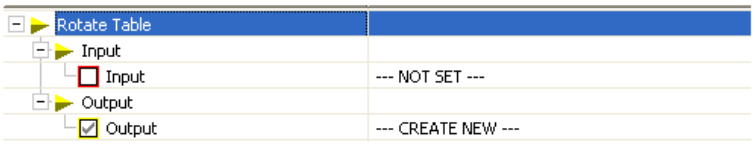
the diagram will also change. Moreover, if you change the order of the rows clicking on its header cell, the diagram will also change.

This is how the above diagram will look like if you order the table using the *Dim02* column.



15.4 Rotating a table

This is not really a table analysis module, but it deals with tables. Use it if you want to rotate a table (that is, put rows as columns and columns as rows). To start it, select the *Tables/Tools/Rotate Table* menu item.

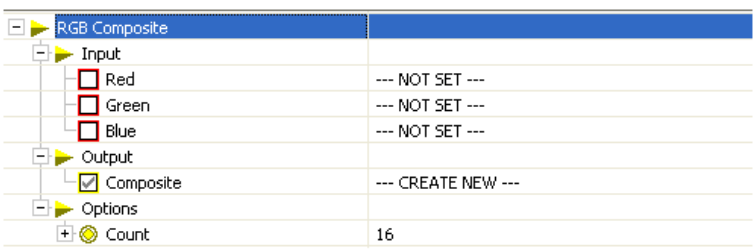


Simply select the input and output tables (which, as usual, can be the same) and press the *OK* button.

15.5 Creating RGB composites

Images are normally composed of three different layers (called channels), each one of them containg a color. If using the usual RGB format, these three channels contain intensity values for red, green and blue (hence the acronym).

If you have this three channels as independent grids, you can join them and create a single full–color image. To do this, use the *RGB Composite* module.



To get the three source layers, you can import an image using the *Import Image* module, setting the *Split Channels* field to true.

Now select each one of this channels in its corresponding field.

The value in the *Count* field is used to create the corresponding color pallete. The number of different classes of the pallete is calculated using the following expression.

$$N = c^n \tag{15.1}$$

Where c is the count value,
and n the number of channels, in this case 3.

15.6 Recreations

Who said that GIS wasn't fun? Apart from being one of the most powerful GISs around, SAGA is, as far as I know, the funniest of them all, since it is the only one that includes recreational stuff.

From games such as the omnipresent minesweeper (who could imagine that this could be implemented within a GIS?!) to more scientifical things such as grids containing Mandelbrot sets, you can find a bunch of very amusing modules in the Recreations.mlb library.

I believe it would not be a good idea not to include some information about this modules in this book but, on the other hand, I find rather useless writing a detailed description of each one of them. Also, exploring this modules by yourself can also be a rewarding experience, so I leave it up to you to discover how they work. Most of them are almost obvious except, perhaps, for the *Newton–Raphson* one.

It is an interactive module which features a capabilities not seen before in any other module: it behaves differently depending on the mouse button you click with. Go and try it yourself... Good luck!

Chapter 16

Programming SAGA modules

16.1 Introduction

Right now you know how to use the basic SAGA functionalities and all its main modules. However, since the range of tasks that can be worked using a GIS is very large, these modules might not be enough for solving some of them. What to do in this case? Well, you can try to find some new module, use another program or, better than that, develop your own module or even a library containing several of them.

What do you need to develop your own modules? First of all you need to use Microsoft Visual C++, the same software used to develop SAGA itself. Unlike SAGA, it is not free software, but you can download the compiler (not the IDE) from the Microsoft website.

No other software is needed except for, of course, SAGA itself, which you will use to debug and test your modules.

I assume that you already have a good command of what has been explained in previous chapters (it doesn't mean that you have to remember exactly how each module works, but you must feel comfortable when using SAGA and not need to go back to the manual each minute), so if you are not sure about that, please read it again and take some more time to practice.

I also assume that you have previously used Visual C++ and you are used to the development environment.

16.2 Modules and libraries

As you already know, modules are grouped in libraries. When creating a module, it has to be contained in a library, and that has to be defined in the source code. Since this is something that you always have to do, I will start explaining the basic structure of a library before writing any line of code.

The guys at the SAGA development team kindly provide a template that you can download from the SAGA website, and I will use it in this chapter. Download the file named MODULE_TEMPLATE.zip. Extract it and you will get the following files:

- Template.dsp
- Template.h
- Template.cpp
- MLB_Interface.cpp

- MLB_Interface.h

This corresponds to a library with just a single module. The structure of the library is defined in the MLB_Interface.cpp file, while its only module is described in the files Template.h and Template.cpp. The file Template.dsp contains a Visual Studio Project file for the whole library.

You can compile this module and you will get a dll file. If you load it in SAGA, a new menu item will appear under the *Modules* menu. This module simply changes the values of all cells in a grid to a fixed value and outputs a new grid, as we will see when we analyse its source code.

We will use this template to create a new library that contains two modules, one of them being a simplified version of the other. The first thing we have to do is to rename the template files. Use the following renaming scheme:

- Template.h → MyModule.h
- Template.cpp → MyModule.cpp
- Template.dsp → MyLibrary.dsp

Of course, renaming the files is not strictly necessary to compile the module library, but it is the first logical step if you want to create your own module...

Do not rename the *MLB_Interface.** files.

We are still missing two files (a .cpp and a header file) corresponding to the second module, that is, the simplified version. We will see later how to add new modules, but first of all we have to make some changes to the MyLibrary.dsp file. Open it with any plain text editor (Windows Notepad is not a powerful one but you can perfectly use it for this purpose...). Replace all *Template* entries with *MyLibrary*, save it, close the editor and now open it with Visual C++ (double clicking on it should automatically launch VC++).ç

Save this file, close the editor, and open MyFirstModule.dsp with Microsoft Visual C++. Using the replace command in VC++, replace all *Template* entries with *MyModule* in MyModule.cpp and MyModule.h.

After this, let's have a look at the source code of the MLB_Interface.cpp file, which defines the structure of a module. Here is the first part of the code (I have removed comments to make it more compact...).

```
extern "C" __declspec( dllexport ) char * Get_Module_Info(int i)
{
  switch( i )
  {
  case MODULE_INTERFACE_INFO_Name: default:
    return( "Template" );

  case MODULE_INTERFACE_INFO_Author:
    return( "SAGA G.b.R. (c) 2003" );

  case MODULE_INTERFACE_INFO_Description:
    return( "Template" );

  case MODULE_INTERFACE_INFO_Version:
    return( "1.0" );
```

```
  case MODULE_INTERFACE_INFO_Menu_Path:
    return( "Template|Template" );
  }
}
```

Here you must define the information about your library. I believe no further comments are needed. The last line of code is used to return the menu path under which all modules contained in the library will be found. If, for example, you use the following line,

```
return( "My First Library|Shape Modules"')
```

all modules will be located under *Modules/My First Library/Shape Module*. Use the slash sign to separate submenus.

After this section, you should include all the headers of your modules. Right now, there is only one include line:

```
#include "Template.h"
```

Since we have renamed the Template.h file, we should change this line. Also, although right now we don't have the cpp and header files for the second module, let's also add an include statement for it. Substitute the above line with the following two ones.

```
#include "MyModule.h"
#include "MyModuleSimpl.h"
```

Each module file defines a C++ class. In the next block of code, these classes have to be specified inside the Create_Module method. The order under which they appear in the *switch* block will determine the order of the menu items under the root submenu defined a few lines above.

```
CModule * Create_Module(int i)
{
  CModule *pModule;

  switch( i )
  {
  case 0:
    pModule = new CTemplate;
    break;

  default:
    pModule = NULL;
    break;
  }

  return( pModule );
}
```

Again, the class of the template has been renamed and we want to add another one corresponding to the simplified version of the first module. Substitute the above block of code with the following one:

```
CModule * Create_Module(int i)
{
  CModule *pModule;

  switch( i )
  {
  case 0:
    pModule = new CMyModule;
    break;

  case 1:
    pModule = new CMyModuleSimpl;
    break;

  default:
    pModule = NULL;
    break;
  }

  return( pModule );
}
```

For each extra module that you want to add to the library, simply add its header and its own entry in the switch block.

Right now, the main structure of the library is defined, so we can start working with the modules themselves and start developing our own algorithms.

The template module (renamed by you to MyModule) is a very good example of module, and contains some of the most important features of SAGA modules along with quite precise comments that are more than enough to understand how to create basic modules. I will try to go a bit beyond this the template structure but not a lot, since the chapter will become quite bulky due to the great complexity of the SAGA API (which can be used completely from any module...). Anyway, don't forget that SAGA modules are also open source, which means that you can read the code of any SAGA module in case you need to do something not explained in here but found in any of this modules. Most of this modules are pretty simple and easy to understand, so don't be frightened to do it. Curiosity is the mother of Science[1].

A basic module (a non-interactive one), contains the following main parts:

- Retrieving information from the user about how the module must be executed
- Execute the module with that information

16.2.1 The constructor. Creating parameters windows

Under these lines you can find the constructor of the class containing the MyModule module (again without comments).

```
CMyModule::CMyModule(void)
{
  Parameters.Set_Name("MyModule");
```

[1] *La curiosidad es la madre de la ciencia*. A Spanish proverb that I translated literally.... Don't know if it makes much sense in english...

```
  Parameters.Set_Description(
    "MyModule\r\n"
    "(created by SAGA Wizard).");

  Parameters.Add_Grid(NULL, "INPUT", "Input" ,
                      "Input for module calculations.", PARAMETER_INPUT);
  Parameters.Add_Grid(NULL, "RESULT", "Output",
                      "Output of module calculations.", PARAMETER_OUTPUT);

  Parameters.Add_Value(NULL, "BOOLEAN", "Boolean",
                        "A value of type boolean.", PARAMETER_TYPE_Bool, true);
  Parameters.Add_Value(NULL, "INTEGER", "Integer",
                       "A value of type integer.", PARAMETER_TYPE_Int, 200);
  Parameters.Add_Value(NULL, "DOUBLE", "Double",
                       "A floating point value.", PARAMETER_TYPE_Double, 3.145);

  Parameters.Add_Select(NULL, "METHOD" , "Method",
                        "Choose a method from this select option.",
                        "First Method\0"
                        "Second Method\0",
                        0 );
}
```

As you can see, all lines make use of the variable `Parameters`, which is defined as a member variable in the CModule class and you can, therefore, access it from every module you create.

The first two lines set the name and the description of the module, and they will be used to label the corresponding menu item and to provide additional information in the module library manager. I believe they are rather self–explaining.

The remaining ones define the parameters window that you will see once you call the module. This module deals only with raster layers and not a lot of additional information is needed, so there are a lot of other different possibilities not covered by the information found in this template module. All of them just use methods from the CParameter class, and will be presented next one by one.

The first thing that you should request from the user is a layer (or a group of them), whether raster or vector. To ask for a raster layer you can use the member function Add_Grid, defined as follows.

```
CParameter * Add_Grid (CParameter *pParent,
                       const char *Ident,
                       const char *Name,
                       const char *Desc,
                       int Constraint);
```

where
`pParent` is a pointer to the parent Parameter (we will see later how to use this to group parameters, but right now you can just set it to `NULL`).
`*Ident` is a module internal identifier string for your parameter

`*Name` is the name of the Parameter displayed to the user
`*Desc` is a string with a detailed description of this parameter displayed to the user
`Constraint` is a descriptor of the input–output characteristic of the demanded Grid, which may have one of the following self explaining values:

- `PARAMETER_INPUT`
- `PARAMETER_OUTPUT`
- `PARAMETER_INPUT_OPTIONAL`
- `PARAMETER_OUTPUT_OPTIONAL`

Here is an example of how to use this, taken straight from the template source code:

```
Parameters.Add_Grid(NULL, "INPUT", "Input" ,
                    "Input for module calculations.", PARAMETER_INPUT);
```

If instead of a grid you need a vector layer, you can use the Add_Shapes method.

```
CParameter * Add_Shapes(CParameter *pParent,
                        const char *Ident,
                        const char *Name,
                        const char *Desc,
                        int Constraint,
                        TShape_Type Shape_Type = SHAPE_TYPE_Undefined);
```

Its structure is quite similar to the Add_Grid method, except for the last optional parameter, which can take the following values:

- `SHAPE_TYPE_Undefined`
- `SHAPE_TYPE_Point`
- `SHAPE_TYPE_Points`
- `SHAPE_TYPE_Line`
- `SHAPE_TYPE_Polygon`

If you need a table, you can use the Add_Table method, which has the same parameters as the Add_Grid one.
In case you need a single value the Add_Value function is the one you need.

Sometimes you might need a group of grids or tables, so using a single selection parameter is not a good idea. To include a multiple selection field in your parameters window, use the following function:

```
CParameter * Add_List(CParameter *pParent,
                      const char *Ident,
                      const char *Name,
                      const char *Desc,
                      int Constraint,
                      TParameter_Type Item_Type,
                      bool bProjectDependent = true);
```

For the Item_Type parameter you can use any of the values of type TParameter_Type defined in the Parameters.h file, though you are only likely to use the following ones:

- PARAMETER_TYPE_Bool
- PARAMETER_TYPE_Char
- PARAMETER_TYPE_Int
- PARAMETER_TYPE_Double
- PARAMETER_TYPE_Grid
- PARAMETER_TYPE_Table
- PARAMETER_TYPE_Shapes

Other less common parameters can be requested using different methods from the CParameters class. Here you have a couple of examples.

```
Parameters.Add_FilePath(NULL,
                        "OUTPUTPATH",
                        "Output Path",
                        "Output Path",
                        "",
                        "",
                        true,
                        true);

Parameters.Add_Value(pNode,
                     "COLORBARS",
                     "Color para graficos de línea",
                     "",
                     PARAMETER_TYPE_Color,
                     RGB(0,0,255));
```

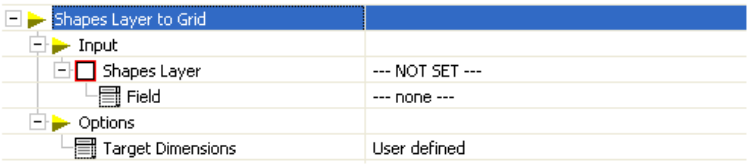
For more information, you can have a look at the file Parameters.h in the Parameters folder under Saga_API.

As you can see, all these methods return a pointer to a CParameter object. Using this pointer as the first parameter when calling another method will cause the corresponding parameter for that function to be placed under the one returned by the first one. Let's see an example. The following code, taken from the *Shapes Layer to Grid* module, will generate the first node of the parameters window shown below.

```
CParameter *pNode_0, *pNode_1;

pNode_0 = Parameters.Add_Shapes(NULL, "INPUT", "Shapes Layer",
                                "",PARAMETER_INPUT );

pNode_1 = Parameters.Add_Select(pNode_0, "FIELD", "Field",
                                  "","--- none ---\0");
```



You can group your parameters adding nodes. To do this, you can use the Add_Node() method. Here is an example of how to use this, taken from one the kriging modules.

```
pNode_0 = Parameters.Add_Node(
          NULL, "PARMS", "Additional Parameters",
          ""
      );

pNode_1 = Parameters.Add_Value(
          pNode_0, "PARM_LIN_B", "Linear Regression",
          "Parameter B for Linear Regression:\r\n   y = Nugget + B * x",
          PARAMETER_TYPE_Double, 1.0
        );
```

16.2.2 The On_Execute() method

Once the parameters window has been defined, it is time now to write the main algorithms that constitute the core of the module.

When the user introduces the required information and presses the *OK* button in the parameters window, the method On_Execute() is called. Here is where you have to put all your ideas and where all the processing has to be done.

However, the first thing that you must do before writing the algorithms themselves is to put the information entered by the user into some handy variables. This information is stored in the Parameters object, and can be accessed using some methods from the CParameters class.

Here is the block of code in the template module:

```
pInput = Parameters("INPUT")->asGrid();
pResult = Parameters("RESULT")->asGrid();

d = Parameters("DOUBLE")->asDouble();
```

To get the value of any parameter, simply use Parameters(*parameter_identifier*) and then one of the following self explaining functions depending on the type of of parameter you are dealing with:

```
bool asBool (void)
```

```
char asChar (void)
int asInt (void)
long asLong (void)
long asColor (void)
double asDouble (void)

void * asPointer (void)

char * asString (void)
SYS_LOGFONT * asFont (void)
CColors * asColors (void)

CDataObject * asDataObject(void)
CGrid * asGrid (void)
CTable * asTable (void)
CShapes * asShapes (void)
```

If you used a multiple selection field, you must use the GetData() method to get a pointer to the list and then use its own methods to get each one of its element. The following example will help you understand how to do this.

```
CParameter_List *pList;

if( (pList = (CParameter_List *)Parameters("LIST_OF_RASTER_LAYERS")->Get_Data()) !=
     NULL && pList->Get_Count() > 0 ){
  for (int i=0; iLayer<pList->Get_Count(); iLayer++){
    if( (pGrid = pList->asShapes(i)) != NULL ){
    //do whatever with the grid
    }
  }
}
```

The Get_Count() method is used to check that the list contains at least one element, and also to go through all the elements. Once you have the CParameter_List object, you can use all the as*Whatever*() methods, using the number of the element that you want to get (ordering is zero–based) as a parameter for the method.

Once all the information entered by the user has been processed, it's time to do something with it. Simply lay your algorithmic ideas using plain C++ code and the methods from the objects that you created from the information provided through the parameters window.

The code in the template.cpp look like this:

```
for(y=0; y<Get_NY() && Progress_Check(y); y++){
  for(x=0; x<Get_NX(); x++){
    pResult->Set_Value(x, y, d * pInput->asDouble(x,y));
  }
}
```

Of course, it is a very simple example, and more complex algorithms can be created, involving other methods, apart from the self–explaining ones found in this code snippet. It's beyond the scope of this chapter to describe all this methods, but you can easily get the necessary information just looking at the files of the SAGA source code where the most important classes (namely CGrid, CShapes, CTable and CModule are described.

16.3 Interactive modules

Creating an interactive module is a bit different, but not more difficult than creating a non–interactive one. The main difference is that you should put the main algorithms not in the OnExecute() method, but in another one named On_Set_Position, which is called each time the user clicks on the grid window.

A good idea is to put the assignment sentences in the OnExecute() method (which is called just once), and then all the algorithms in the On_Set_Position one.

The coordinates("real" or grid) of the point where the user has clicked can be retrieved using the following methods:

- Get_xWorld()
- Get_yWorld()
- Get_xGrid()
- Get_xGrid()

16.4 Calling other modules

You can using otrher modules within your own module, just calling them and passing them the right parameters. To see how this works, we will create a simplified version of the template module, in which the parameter named `DOUBLE` (which was assigned to a variable named `d`) takes a fixed value. Instead of rewriting the module again, we can just make a new one which prompts the user for all the required paramters except this one, and then call the original module passing it a fixed value.

I suposse that right now you can write all the code regarding the parameters window by yourself, so I will just show you what the OnExecute() method looks like in this simplified version. I have used a fixed value of 5 for the `DOUBLE` parameter

```
CMorphometry* pM = new CMorphometry;
pTemplate->Get_Default_Parameters()->Get_Parameter("INPUT")->Set_Value(pInput);
pTemplate->Get_Default_Parameters()->Get_Parameter("OUTPUT")->Set_Value(pOutput);
pTemplate->Get_Default_Parameters()->Get_Parameter("DOUBLE")->Set_Value(5);
pTemplate->Execute(Get_Callback());
delete(pTemplate);
```

You just have to create an object of the class that contains the module you want to execute, supply the parameters it needs and execute it, that's all!! pretty simple and with less than 10 lines of code...

As another example, imagine that you want to calculate the slope and aspect of the new grid you created. You can call the morphometry model as shown in the following code snippet.

```
CMorphometry* pM = new CMorphometry;
pM->Get_Default_Parameters()->Get_Parameter("ELEVATION")->Set_Value(pInput);
pM->Get_Default_Parameters()->Get_Parameter("SLOPE")->Set_Value(pSlopeGrid);
pM->Get_Default_Parameters()->Get_Parameter("ASPECT")->Set_Value(pAspectGrid);
pM->Execute(Get_Callback());
delete(pM);
```

Of course, your have to define pSlopeGrid and`pAspectGrid` as pointers to a `CGrid` before using them as input of the morphometry module.

Bibliography

[1] Al–Smadi, M. *Incorporating Spatial and Temporal Variation of Watershed Response in a GIS–Based Hydrologic Model*. Virginia Polytechnic Institute and State University, 184 págs, 1998

Can be downloaded freely, along with [95] and [50], from *http://scholar.lib.vt.edu/theses/*

[2] Band, L.E. Extraction of channel networks and topographic parameters from digital elevation data, in Beven, K.J.; Kirkby, M.J. *Channel network hydrology*. John Wiley and Sons. 1993.

[3] Band, L.E. Topographic partition of watersheds with digital elevation models, *Wat. Resour. Res.*, 22(1):15–24. 1986

[4] Band L.E. Distributed parameterization of complex terrain. *Surveys in Geophysics* 12: 249–270, 1993

[5] Bao, J. Using GIS for Hydrologic Data–Processing and Modeling in Texas. CRWR Online Report 97-4. Center for research in water resources. 131 pp., 1997

Available at *http://www.ce.utexas.edu/centers/crwr/reports/online.html*

[6] Bauer, J.; Rohdenburg, H.; Bork, H.-R. Ein Digitales Reliefmodell als Vorraussetzung fuer ein deterministisches Modell der Wasser- und Stoff-Fluess, Landschaftsgenese und Landschaftsoekologie, H.10, Parameteraufbereitung fuer deterministische Gebiets-Wassermodelle, *Grundlagenarbeiten zu Analyse von Agrar-Oekosystemen*, (Eds.: Bork, H.-R.; Rohdenburg, H.), p.1–15, 1985

[7] Berry, J.K. Fundamental operations in computer–assisted map analysis, *International Journal of Geographic Information Systems*, 2, 119–136. 1987

[8] Beven, K.J.; M.J. Kirkby, A physically based, variable contributing area model of basin hydrology, *Hydrol. Sci. Bull., 24*, 43-69, 1979.

Bertolo, F *Catchment delineation and characterisation: A review*, EC–JRC, Space Applications Institute, (EUR 19563 EN) Ispra (VA), Italy, 36 págs, 2000

Download it free from *http://agrienv.jrc.it/publications/pdfs/CatchRev.pdf*

[9]

[10] Beven, K.J. and Moore, I.D. (eds.) *Terrain Analysis and Distributed Modelling in Hydrology*, John Wiley and Sons, Chichester, UK, 7–34.

[11] Beven, K.J., Kirkby; M.J.; Schoffield, N.; Tagg, A. Testing a Physically–based Flood Forecasting Model (TOPMODEL) for Three UK Catchments, *Journal of Hydrology*, 69, 119–143, 1984 1995

[12]

[13] Blaszczynski, J. Landform characterization with geographic information systems, *Photogrammetric Engineering and Remote Sensing*, Vol. 63, No. 2, pp. 183–191. 1997

[14] Bonham–Carter, G.F. *Geographic Information Systems for Geoscientists: Modeling with GIS*, Kidlington, Elsevier, 398 págs. 1994

[15] Brabyn, L. GIS Analysis of Macro Landform *Proceedings of The 10th Annual Colloquium of the Spatial Information Research Centre. SIRC 98*. 1998

[16] Brabyn, L. Classification of macro landforms using GIS. *ITC Journal* 97: 26–40, 1997

[17] Burrough, P.A.; McDonnell, R.A. *Principles of Geographical Information Systems*. Oxford University Press. 333 pp. 1998

[18] Burt, T.P.; Butcher, D.P. Topographic controls of soil moisture distributions. *Journal of Soil Science* 36: 469–486, 1986

[19] Carrara, A.; Bitelli, G.; Carla, R. Comparison of techniques for generating digital terrain models from contour lines. *International Journal of Geographical Information Systems*. Vol. 11, no. 5, pp. 451–473. 1997

[20] Carter, J.R. Digital representations of topographic surfaces. *Photogrammetric Engineering and Remote Sensing* 54: 1577–1580, 1988

[21] Chairat, S. y Delleur J. W. Effects of the topographic index distribution on predicted runoff using GRASS. *Water Resources Bulletin* 29: 1029–1034. 1993

[22] Chang, K.T.; Tsai, B.W. The effect of DEM resolution on slope and aspect mapping. *Cartography and Geographic Information Systems*, Vol. 18, no. 1, pp. 69–77. 1991

[23] Charleux-Demargne J. *Qualité des Modèles Numériques de Terrain pour l'Hydrologie — Application à la Caractérisation du Régime de Crues des Bassins Versants —*. Thèse, Université de Marne–la–Vallée, 275 p. 2001

Can be downloaded from*http://www.montpellier.cemagref.fr/doc/publications/theses/julie-charleux--demargne.html*

[24] Chen, H. *Object Watershed Link Simulation (OWLS)*. PhD Dissertation, Oregon State University. 1996

Source code and text downloadable from *http://www.hydromodel.com*

[25] Chen, H.; Beschta, R.Dynamic Hydrologic Simulation of the Bear Brook Watershed in Maine (BBWM). *Environmental Monitoring and Assessment* 55:53–96, 1999

[26] Chorowicz, J.; Ichoku, C.; Riazanoff, S.; Kim, Y.; Cervelle, B. A combined algorithms for automated drainage network extraction, *Wat. Resour. Res.*, 28(5): 1293–1302. 1992

[27] Chorowicz, J.; Kim, J;Manoussis, S.; Rudant, J.P.; Foin P. et al A new technique for recognition of geological and geomorphological patterns in digital terrain models, *Remote. Sens. Environ.* Vol. 29, pp. 229-239., 1989

Clark, I; Harper, W.H. *Practical Geostatistics 2000* Ecosse North America Llc, Columbus, Ohio, 442 págs., 2000

[28] Collins, S.H.M; Moon, G.C. Algorithms for dense digital terrain models. *Photogrammetric Engineering and Remote Sensing* 47: 71–76, 1981

[29] Collins F.C.; Bolstad, P.V. A comparison of spatial interpolation techniques in temperature estimation. *Proceedings of Third International Conference/Workshop on Integrating GIS and Environmental Modelling* CD-ROM. Santa Fe, New Mexico, USA, 1996.

Available at *http://www.ncgia.ucsb.edu/conf/SANTA_FE_CD-ROM/sf_papers/collins_fred/collins.html*

[30] Conrad, O. Derivation of Hydrologically Significant Parameters from Digital Terrain Models. PhD Thesis. Dept. for Physical Geography, University of Göttingen, 1998

Original thesis from the autor of SAGA and DiGeM. Can be downloaded (in German) from*http://www.-geogr.uni–goettingen.de/pg/saga/digem/index.html*

[31] Costa-Cabral, M. C.; Burges,S. J. Digital elevation model networks (DEMON): A model of flow over hillslopes for computation of contributing and dispersal areas. *Wat. Resour. Res.* 30: 1681–92. (1994)

[32] Cowen, J. A proposed method for calculating the LS factor for use with the USLE in a grid–based environment. *Proceedings of the thirteenth annual ESRI user conference*, pp. 65–74. 1993 Cressie, N.A.C. *Statistics for Spatial Data, Revised Edition*, Wiley Series in Probability and Mathematical Statistics. New York, Wiley, 1993

[33] Desmet, P.J.J.; Govers, G. Comparison of routing algorithms for digital elevation models and their implications for predicting ephemeral gullies. *International Journal of Geographical Information Systems* 10: 311–331, 1996

[34] Desmet, P.J.J.; Govers, G. A GIS procedure for automatically calculating the USLE LS factor on topographically complex landscape units. *Journal of Soil and Water Conservation* 51: 427–433, 1996

[35] Desmet, P.J.J.; Govers, G. Comment on "Modelling topographic potential for erosion and deposition using GIS". *International Journal of Geographical Information Science* 11: 603–610, 1997

[36] Deursen, W.P.A.; Heil, G.W.; *PCRaster*. Department of Physical Geography, Utrecht University, Utrecht, The Netherlands. 1995

Some interesting technical ideas an be found at the PCRaster website at *http://www.pcraster.nl/*

[37] Dietrich, W.E., Wilson, C.J.; Montgomery, D.R.; McKean, J. Analysis of erosion thresholds, channel networks, and landscape morphology using a digital terrain model, *J. Geol., 101*, 259–278, 1993.

[38] Dikau, R. The application of a digital relief model to landform analysis in geomorphology. In Raper, J.(ed.) *Three dimensional application in Geographic Information Systems*, pp 51-77. Taylor and Francis. 1989

[39] Dikau, R. Computergestützte geomorphographie und ihre anwendung in der regionalisierung des reliefs. *Petermanns Geographische Mitteilungen*, 138:99-114. 1994

[40] Drake, N.A.; Vafeidis, A.; Wainwright, J.; and Zhang, X.; Modelling soil erosion using remote sensing and GIS techniques, *Proceedings of RSS 95 Remote Sensing in Action, 11-14 September 1995, Southampton*, 217–224. 1995

[41] Duan, J.; Miller, N. A generalized power function for the subsurface transmissivity profile in TOPMODEL, *Wat. Resour. Res.* 33 (11) 2559–2562. 1997

[42] Dubayah, R.; Rich, P.M. Topographic solar radiation models for GIS. *International Journal of Geographical Information Systems* 9: 405–419, 1995

[43] Dubin, A. M.; Lettenmaier, D.P. *Assessing the Influence of Digital Elevation Model Resolution on Hydrologic Modeling*, Water Resources Series, Technical Report 159, University of Washington, Seattle. 1999

[44] Dunn, M.; Hickey, R. The effect of slope algorithms on slope estimates within a GIS. *Cartography*, Vol. 27, no. 1, pp. 9–15 1998

[45] Dymond, J.R.; Derose, R.C.; Harmsworth, G.R. Automated mapping of land components from digital elevation data. *Earth Surface Processes and Landforms* 20: 131–137, 1995

[46] Dymond, J. R.; Harmsworth, G.R. Towards automated land resource mapping using digital terrain models, *ITC Journal*, Vol. 2, pp. 129–138. 1994

[47] Evans, I. S., General geomorphometry, derivatives of altitude, and descriptive statistics. En Chorley, R. J. (ed.) *Spatial Analysis in Geomorphology*, Methuen, London. pp.17-90. 1972

[48] Evans, I.S. An integrated system of terrain analysis and slope mapping, *Zeitschrift für Geomorphologie, N.F. Supplementband*, 36, 274–295. 1980

[49] Fairfield, J.; Leymarie P. Drainage networks from grid digital elevation models. *Wat. Resour. Res.* 27(5): 709–717, 1991

[50] Fedak, R. *Effect of Spatial Scale on Hydrologic Modeling in a Headwater Catchment.* Virginia Polytechnic Institute and State University, 179 págs. 1999

[51] Felicísimo, A. M. *Modelos Digitales del Terreno. Introducción y aplicaciones en la Ciencias ambientales.* Oviedo, Pentalfa, 222 págs. 1994

Great reference about terrain analysis. Can be downoaded (in Spanish) from *http://www.etsimo.uniovi.-es/f̃eli/*

[52] Florinski, I. V. Combined analysis of digital terrain models and remotely sensed data in landscape investigations, *Prog. Phys. Geogr.*, 22(1), 33–60, 1998.

[53] Florinsky, I. Accuracy of local topographic variables derived from digital elevation models. *International Journal of Geographical Information Science*, Vol. 12, no. 1, pp. 47–61. 1998

[54] Fortin, J.P., Moussa R.; Bocquillon C.; Villeneuve, J.P. Hydrotel, un modèle hydrologique distribué pouvant bénéficier des données fournies par la télédétection et les systèmes d′information géographique. *Revue des Sciences de l´Eau*, 8 : 97–124. 1995

Available at *http://www.inrs-eau.uquebec.ca/activites/modeles/hydrotel/fr/accueil.htm*

[55] Frances, F.; Benito, J. La modelacion distribuida con pocos par'ametros de las crecidas. *Ingenier'ia del agua*. Vol. 2 N° 4, pp. 7–24, 1995

[56] Franklin, S. Geomorphometric processing of digital elevation models, *Computers and Geosciences*, Vol. 13, No. 6, pp. 603–609. 1987

[57] Freeman, T.G. Calculating catchment area with divergent flow based on a regular grid, *Computers and Geosciences*, 17(3): 413–422. 1991

[58] Forgy, E. Cluster Analysis of multivariate data: efficiency vs. interpretability of classifications, *Biometrics* 21:768, 1965

[59] Gao, J. Resolution and accuracy of terrain representation by grid DEMs at a micro–scale. *International Journal of Geographical Information Science*. Vol. 11, no. 2, pp. 199–212. 1997

[60] Garbrecht, J,; Martz L.W. Grid size dependency of parameters extracted from digital elevation models. *Computers and Geosciences* 20: 85–7. 1994

[61] Garbrecht, J.; Martz, L. W. Comment on "A Combined Algorithm for Automated Drainage Network Extraction" by J. Chorowicz, C. Ichoku, S. Riazanoff, Y. J. Kim, and B. Cervelle, *Wat. Resour. Res.*, 29(2):535–536, 1993.

[62] Garbrecht, J.; Martz L.W. Network and Subwatershed Parameters Extracted From Digital Elevation Models: The Bills Creek Experience. *Water Resources Bulletin*, American Water Resources Association, 29(6):909–916, 1993.

[63] Garbrecht, J.; Martz L.W. y Starks, P.J. Automated Watershed Parameterization from Digital Landscapes: Capabilities and Limitations. *Proceedings of 14th Annual American Geophysical Union Front Range Branch Hydrology Days*, Colorado State University, Fort Collins, Colorado, pp. 123–134, 1994.

[64] Garbrecht, J.; Martz L.W. Digital Landscape Parameterization for Hydrologic Applications. *Proceedings of HydroGIS '96, International Conference on Application of Geographic Information Systems in Hydrology and Water Resources Management*, Vienna, Austria, IAHS Publication No. 235, pp. 169–173, 1996.

[65] Garbrecht, J.; Martz L.W. Comment on "Digital Elevation Model Grid Size, Landscape Representation, and Hydrologic Simulation" by Weihua Zhang and David R. Montgomery. *Wat. Resour. Res.*, 32(5):1461–1462, 1996.

[66] Garbrecht, J.; Martz L.W. The Assignment of Drainage Direction over Flat Surfaces in Raster Digital Elevation Models. *Journal of Hydrology*, 193:204–213. 1997

[67] Garbrecht, J.; Martz L.W. Automated Channel Ordering and Node Indexing for Raster Channel Networks. it Computers and Geosciences, 23(9): 961–966. 1997

[68] Goovarets, P. Performance comparison of geostatistical algorithms for incorporating elevation into the mapping of precipitation. *Journal of Hydrology* 228:113–129. 2000

On–line version can be downloaded from *http://www.geovista.psu.edu/sites/geocomp99/Gc99/023/-gc_023.htm*

[69] Gousie, M; Franklin, R. Converting Elevation Contours to a Grid. *Proceedings of the Eighth International Symposium on Spatial Data Handling.* pp. 647–656. 1998

Can be downloaded from *http://cs.wheatonma.edu/mgousie/*

[70] Gousie, M. *Contours to Digital Elevation Models: Grid–Based Surface Reconstruction Methods.* PhD thesis, Rensselaer Polytechnic Institute, 1998.

[71] Gousie, M.; Franklin, W. R. Constructing a DEM from Grid–based Data by Computing Intermediate Contours. *GIS 2003: Proceedings of the Eleventh ACM International Symposium on Advances in Geographic Information Systems* pp. 71–77, New Orleans, 2003

[72] Goward, S., Markham, B., Dye, D., Dulaney, W., and Yang., J. Normalized difference vegetation index measurements from the Advanced Very High Resolution Radiometer, *Remote Sens. Environ.* 35:257-277, 1991

[73] Gutman, G. Vegetation indices from AVHRR: an update and future prospects,*Remote Sens. Environ.*, 35:121-136, 1991

[74] Gyasi–Agyei, Y., G. Willgoose, and F. P. De Troch, Effects of vertical resolution and map scale of digital elevation models on geomorphological parameters used in hydrology, Hydrol. Processes, 9, 363–382, 1995.

[75] Hammer, R.D.; Young, F.J.; Wollenhaupt, N.C.; Barney, T.L; Haithcoate, T.W. Slope class maps from soil survey and digital elevation models, *Soil Science Society of America Journal*, 59(2), 509–519. 1995.

[76] Helmlinger, K.R.; Kumar, P.; Foufoula–Georgiou, E. On the use of digital elevation model data for Hortonian and fractal analyses of channel networks, *Wat. Resour. Res.*, 29(8), 2599–2613. 1993

[77] Hennrich, K.; Schmidt, J.; Dikau, R. *Regionalization of geomorphometric parameters in hydrologic modelling using GIS.* IAHS Publications.

[78] Hickey, R.; Smith, A.; Jankowski, P. Slope Length Calculations from a DEM within Arc/Info GRID, *Computing, Environment and Urban Systems*, Vol. 18, No. 5, pp. 365–380. 1994

[79] Hickey, R. Slope Angle and Slope Length Solutions for GIS. *Cartography*, Vol. 29, no. 1, pp. 1–8. 2000

[80] Hilditch, C.J., Comparison of thinning algorithms on a parallel processor, *Image Vision Comput.*, vol. 1, no. 3, pp. 115-132, 1983.

[81] Hjelmfelt, A. T., Jr. Fractals and the river–length catchment–area ratio, *Wat. Resour. Bull.*, 24(2), 455–459, 1988.

[82] Holmgren, P. Multiple flow direction algorithms for runoff modelling in grid based elevation models: an empirical evaluation, *Hydrological Processes*, 8: 327–334. 1994

[83] Horn, B.K.P. Hill shading and the reflectance map, Proceedings of the I.E.E.E., 69, 14. 1981

[84] Horritt, M. S.; Bates, P. D. Effects of spatial resolution on a raster based model of flood flow. *Journal of Hydrology.* 253, 239–249. 2001

[85] Horton, R.E., Erosional development of streams and their drainage basins: Hydrophysical approach to quantitative morphology, *Bull. Geol. Soc. Am., 56*, 275–370, 1945.

[86] Hutchinson, M.F. A new procedure for gridding elevation and stream line data with automatic removal of spurious pits. *Journal of Hydrology*, 106, 211–232. 1988.

[87] Hutchinson, M. F. Interpolating mean rainfall using thin plate smoothing splines. *International Journal of Geographical Information Systems* 9: 385–403. 1995

[88] Jenson, S.K.; Domingue, J.O. Extracting topographic structure from digital elevation model data for geographic information system analysis. *Photogrammetric Engineering and Remote Sensing* 54: 1593–1600, 1988

[89] Jenson, S. K. Applications of hydrologic information automatically extracted from Digital Elevation Model. *Hydrological Processes* Vol. 5, Issue No. 1, pp. 31–44. 1991

[90] Johnson R.D.; Kasischke, E.S. Change vector analysis: a technique for the multispectral monitoring of land cover and condition. *Int. J. Remote Sensing*, vol. 19, no. 3, 411-426, 1998.

[91] Jones, J.A.A. The initiation of natural drainage networks. *Progress in Physical Geography* 11: 205–245, 1987

[92] Jones, K.H. A comparison of eight algorithms used to compute slopes as a local property of the DEM, *Proceedings of the GIS Research UK 1996 Conference*, 7–12. 1996.

[93] Julien, P. Y.; Saghafian, B.; Ogden, F. L. Raster–based hydrologic modeling of spatially varied surface runoff. *Water Resources Bulletin* 31: 523–536. 1995

[94] Kahn, K.N. a Geographic Information System based spatially distributed rainfall-runoff model. M.S Thesis. University of Pittsburgh.

Available at *http://etd.library.pitt.edu/ETD/available/etd-02082002-171103*

[95] Kilgore, J. *Development and Evaluation of a GIS-Based Spatially Distributed Unit Hydrograph Model.* Virginia Polytechnic Institute and State University, 1997

[96] Kumar, L.; Skidmore, A.K.; Knowles, E. Modelling topographic variation in solar radiation in a GIS environment. *International Journal of Geographical Information Science* 11: 475-97, 1997

[97] Lammers, R. B.; Band, L. E. Automating object representation of drainage basins. *Computers and Geosciences*, 16, 787–810. 1990

[98] Lanfear, K.J. A fast Algorithm for Automatically Computing Strahler Stream Order. *Water Resources Bulletin*, Vol. 26, Num 6, pp. 977–981, 1990

[99] Lea, N. L. An aspect driven kinematic routing algorithm. En Parsons, A. J.; Abrahams, A. D. *Overland Flow: Hydraulics and Erosion Mechanics*, New York, Chapman & Hill. 1992

[100] Liang, C.; Mackay, D.S. A general model of watershed extraction and representation using globally optimal flow paths and up–slope contributing areas. *International Journal of Geographical Information Science*, 14(4), 337–358, 2000 Available along with [102] at *http://water.geog.buffalo.edu/ehmg/pubs.html*

[101] Lynch, S.D. Converting Point Estimates of Daily Rainfall onto a Rectangular Grid. *Proceedings of the ESRI User Conference 98*. 1999.

[102] Mackay, D. S. y Band, L. E. Extraction and representation of nested catchment areas from digital elevation models in lake–dominated topography. *Water Resour. Res.* 34: 897–901. 1998

[103] Martínez, V.; Dal–Ré, R.; García, A.I.; Ayuga, F.; Modelaci'on distribuida de la escorrent'ia superficial en peque'nas cuencas mediante SIG. Evaluaci'on experimental, *Ingenieria Civil* No 117, CEDEX Centro de Estudios de Técnicas Aplicadas, 2000;

[104] Martz, L. W. y Garbrecht, J. Numerical Definition of Drainage Network and Subcatchment Areas from Digital Elevation Models. *Computers and Geosciences*, 18(6):747–761, 1992.

[105] Martz, L. W. y Garbrecht, J. DEDNM: A Software System for the Automated Extraction of Channel Network and Watershed Data from Raster Digital Elevation Models. *Proceedings of the Symposium on Geographic Information Systems in Water Resources*, J. M. Harlin and K. J. Lanfear (Eds.), American Water Resources Association, Mobile, Alabama, pp. 211–220, 1993.

[106] Martz, L. W. y Garbrecht, J. Automated Extraction of Drainage Network and Watershed Data from Digital Elevation Models. *Water Resources Bulletin*, American Water Resources Association, 29(6):901–908, 1993.

[107] Martz, L. W. y Garbrecht J. Comment on "Automated Recognition of Valley Lines and Drainage Networks From Grid Digital Elevation Models: A Review and a New Method" by A. Tribe. *Journal of Hydrology*, 167(1):393–396, 1995.

[108] Martz, L.W.; de Jong, E. Catch: A FORTRAN program for measuring catchment area from digital elevation models, *Computers and Geosciences*, 14(5): 627–640. 1988

[109] Martz, L.W. and Garbrecht, J. An outlet breaching algorithm for the treatment of closed depressions in a raster DEM. *Computers and Geosciences* 25, 835–844. 1999

[110] Martz, L.W.; Garbrecht, J.The treatment of flat areas and closed depressions in automated drainage analysis of raster digital elevation models. *Hydrological Processes* 12, 843–855. 1998

[111] Mark, D.M. Automated detection of drainage networks from digital elevation models, *Cartographica*, 21(2/3): 168–178. 1984

[112] Marks, D.; Dozier, J.; Frew, J. Automated Basin Delineation From Digital Elevation Data. *Geo. Processing*, 2: 299–311. 1984

[113] McCool, D.K.; Foster, G.R.; Mutchler, C.K.; Meyer, L.D. Revised slope length factor for the Universal Soil Loss Equation. *Trans. ASAE*, 32, 1571–1576. 1989

[114] Mitasova, H.; Hofierka, J. ; Zlocha, M. y Iverson, L. R. Modeling topographic potential for erosion and deposition using GIS. *International Journal of Geographical Information Systems 10*: 629–41. 1996

[115] Mitasova, H.; Mitas, L. Multiscale soil erosion simulations for land use management. En Harmon, R.; Doe, W. (eds.) *Landscape erosion and landscape evolution modeling.* Kluwer Academic/Plenum Publishers, pp. 321–347. 2001

[116] Montgomery, D.R.; W.E. Dietrich, Where do channels begin?, *Nature, 336*, 232–234, 1988.

[117] Montgomery, D.R.; W.E. Dietrich, Source areas, drainage density, and channel initiation, *Water Resour. Res., 25*, 1907–1918, 1989.

[118] Montgomery, D.R.; W.E. Dietrich, Channel initiation and the problem of landscape scale, *Science, 255*, 826–830, 1992.

[119] Montgomery, D.R.; Foufoula–Georgiou, E. Channel network source representation using digital elevation models, *Wat, Resour. Res.*, 29(12), 3925–3934. 1993.

[120] Monmonier, M.S. Measures of Pattern Complexity for Choropleth Maps, *The American Cartographer*, 1, 2, 159-169. 1974.

[121] Moore, I.D.; Burch, G.J. Sediment transport capacity of sheet and rill flow: Application of unit stream power theory. *Wat. Resour. Res.* 22: 1350–1356, 1986

[122] Moore, I.D. and Burch, G.J. Physical basis of the length–slope factor in the Universal Soil Loss Equation. *Soil Science Society of America Journal* 50: 1294–1298, 1986

[123] Moore, I.D.; Burch, G.J. Modelling erosion and deposition: Topographic effects. *Transactions of the American Society of Agricultural Engineers* 29: 1624–1630, 1640. 1986

[124] Moore, I.D.; Nieber, J.L. Landscape assessment of soil erosion and non–point source pollution. *Journal of the Minnesota Academy of Science* 55: 18–25. 1991

[125] Moore, I.D. and Wilson, J.P. Length–slope factors for the Revised Universal Soil Loss Equation: Simplified method of estimation. *Journal of Soil and Water Conservation* 47: 423–428. 1992

[126] Moore, I.D.; Lewis, A.; Gallant, J.C. Terrain attributes: Estimation methods and scale effects. En Jakeman, A.J.; Beck, M.B.; McAleer, M.J. (eds.) *Modelling Change in Environmental Systems.* New York, NY, John Wiley and Sons: 189–214, 1993

[127] Moore, I. D.; Grayson, R. B.; Ladson, A. R. Digital terrain modelling: a review of hydrological, geomorphological, and biological applications. *Hydrological Processes*, 5(3):3–30. 1991

[128] Moore, I.D.; Wilson, J.P. Length–slope factors for the Revised Universal Soil Loss Equation: Simplified method of estimation. *J. Soil and Water Cons.* 47(5):423–428. 1992

[129] Morillo, J.; Pozo, J.; Pérez, F.; Rodríguez, M.C.; Rebollo, F.J. An'alisis de calidad de un modelo digital de elevaciones generado con distintas t'ecnicas de interpolaci'on. *Actas del XIV Congreso Internacional de Ingeniería Gráfica* Santander, España, 2002

[130] Murphy, D.L. Estimating Neighborhood Variability with a Binary Comparison Matrix, *Photogrammetric Engineering and Remote Sensing*, 51, 6, 667-674. 1985.

[131] O'Callaghan, J. F. y Mark D.M. The extraction of drainage networks from digital elevation data. *Computer Vision, Graphics and Image Processing* 28: 323–44. 1984

[132] Olaya, V. *Integraci'on de modelos computacionales geomorfol'ogicos hidrologicos y selvicolas para el desarrollo de soluciones SIG espec'ificas en hidrolog'ia forestal de peque'nas y medianas cuencas vertientes.* Proyecto Fin de Carrera, Universidad Polit'ecnica de Madrid, Madrid, 813 págs, 2002

[133] . Perry, C.R. Jr.; L.F. Lautenschlager. Functional equivalence of spectral vegetation indices. *Remote Sens. Environ.* 14:169-182, 1984.

[134] Peuker, T.K.; Douglas, D.H. Detection of surface–specific points by parallel processing of discrete terrain elevation data, *Computer Graphics and Image Processing*, Vol.4, No.4, 375–387, 1975

[135] Pilesjo, P.; Zhou, Q., A multiple flow direction algorithm and its use for hydrological modelling, *Geoinformatics'96 Proceedings*, 26-28 Abril, West Palm Beach, FL, 2: 366–376. 1996

[136] Pilesjo, P.; Zhou, Q.; Harrie, L. Estimating flow distribution over digital elevation models using a form-based algorithm. *Geographical Information Sciences*, 4(1-2), pp 44–51. 1998

[137] Pilotti, M.; Gandolfi, C.; Bischetti, G.B. Identification and analysis of natural channel networks from digital elevation models, *Earth Surface Processes and Landforms*, 21: 1007–1020. 1996

[138] Planchon, O.; Darboux, F. A fast, simple and versatile algorithm to fill the depressions of digital elevation models, *Catena*, Vol. 46, pp. 159–176, 2001

[139] Qian, J.; Ehrich, R.W.; Campbell, J.B., DNESYS — An expert system for automatic extraction of drainage networks from digital elevation data —, *IEEE Transactions on Geoscience and Remote Sensing*, 28(1): 29–45. 1996

[140] Quinn, P.F.; Beven, K.J.;Lamb, R.; The $\ln(a/\tan\beta)$ index: how to calculate it and how to use it within the TOPMODEL framework, *Hydrological Processes*, 9:161–182. 1995

[141] Quinn, P.F.; Beven, K.J.; Chevallier, P.; Planchon, O.; The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models, *Hydrological Processes*, 5: 59–79. 1991

[142] Richardson, A. J. & Wiegand, C. L. Distinguishing vegetation from soil background information. *Photogr. E. R.* 43: 1541-1552. 1977.

[143] Rieger, W., Automated river line and catchment area extraction from DEM data, *Proceedings of 17th Congress of ISPRS, 2-14 August, Washington, D.C.*, B4: 642–649. 1992

[144] Robinson, G. J. The accuracy of digital elevation models derived from digitised contour data, *Photogramm. Rec.*, 14(83), 805–814, 1994

[145] Rubin, J. Optimal Classification into Groups: An Approach for Solving the Taxonomy Problem, *J. Theoretical Biology*, 15:103-144, 1967

[146] Saghafian, B.; Julien, P.Y.; Rajaie, H. Runoff hydrograph simulation based on time variable isochrone technique *Journal of Hydrology*, 261 (1–4) pp. 193-203, 2002

[147] Srivastava A. *Comparison of two algorithms for removing depressions and delineating flow networks for grid digital elevation models*. Virginia Polytechnic Institute and State University, 132 págs. 2000

[148] Star, J,; Estes, J, *Geographic Information Systems*. Prentice Hall, Englewood Cliffs, New Jersey, 1990

[149] Strahler, A. N. Quantitative analysis of watershed geomorphology. *EOS Trans. Agu.* 38. 912–920

[150] Tabios G.Q.M; Salas, J.D. A comparative analysis of techniques for spatial interpolation of precipitation. *Water Resour. Bull.*, 365–380. 1985 bibitemtarboton4 Tarboton, D. G. A new method for the determination of flow directions and upslope areas in grid digital elevation models. *Water Resour. Res. 33*: 309–319. 1997

The original article about the D∞ method. Can be downloaded from *http://www.engineering.usu.edu/-cee/faculty/dtarb/* along with other articles by the same author.

[151] Tarboton, D.G.; Bras, R.L.; Rodriguez—Iturbe, I. On the extraction of channel networks from digital elevation data. *Hydrological Processes*, Vol.5, 81–100, 1991

[152] Tarboton, D.G.; Shankar, U. (1998), The Identification and Mapping of Flow Networks from Digital Elevation Data, *Invited Presentation at AGU Fall Meeting, San Francisco*, 1998

[153] Tarboton, D.G.; Ames, D. P. Advances in the mapping of flow networks from digital elevation data, *World Water and Environmental Resources Congress*, Orlando, Florida, ASCE. 2001

[154] Tarboton, D.G. Terrain Analysis Using Digital Elevation Models in Hydrology, *23rd ESRI International Users Conference*, San Diego, California, 2003

[155] Tribe, A. Automated recognition of valley lines and drainage networks from grid digital elevation models: a review and a new method. *Journal of Hydrology*, 139: 263–293. 1992

[156] Tribe, A. Automated recognition of valley heads from digital elevation data, *Earth Surface Processes and Landforms*, Vol.16, 33–49, 1991

[157] Tribe, A. Towards the automated recognition of landforms (valley heads) from digital elevation models, *Proc. of the 4th Intern. Symposium on Spatial Data Handling*, 45–52, 1990

[158] Turcotte, R.; Fortin, J.–P.; Rousseau, A.N.; Massicotte, S.; Villeneuve, J.–P. Determination of the drainage structure of a watershed using a digital elevation model and a digital river and lake network. *Journal of Hydrology*, 240(3-4):225-242. 2001

[159] Turner, M.G. Landscape Ecology: The Effect of Pattern on Process, *Annu. Rev. Ecol. Syst.*, 20, 171-197. 1989.

[160] United States Geological Survey. *Digital Elevation Models: Data Users Guide*. Reston, VA, United States Geological Survey, 1993

[161] United States Geological Survey. *A Bibliography of Terrain Modeling (Geomorphometry), the Quantitative Representation of Topography — Supplement 4*. United States Geological Survey, 157 págs., 2001

A huge collection of more than 1600 references. Available at *http://geopubs.wr.usgs.gov/open-file/of02-465/*

[162] Van Remortel, R.; Hamilton, R.; Hickey, R. Estimating the LS factor for RUSLE through iterative slope length processing of digital elevation data. *Cartography*, Vol. 30, no. 1, pp. 27–35. 2001

[163] Walker, J.P.; Willgoose, G.R. On the effect of DEM accuracy on hydrology and geomorphology models, *Wat. Resour. Res.*, 35(7), 2259–2268. 1998

[164] Wilson, J.P.; Gallant, J.C. (eds.) *Terrain Analysis: Principles and Applications*. New York, NY, John Wiley and Sons, 2000

[165] Wilson, J.P. Estimating the topographic factor in the universal soil loss equation for watersheds. *Journal of Soil and Water Conservation*, 41, 179–184. 1986

[166] Wise, S.M. The effect of GIS interpolation errors on the use of DEMs in geomorphology. En Lane, S.N.; Richards, K.S.; Chandler, J.H. (eds.) *Landform Monitoring, Modelling and Analysis*. Wiley, Chichester. 139-164, 1998

[167] Wise S.M. Assessing the quality for hyrdological applications of digital elevation models derived from contours. *Hydrological Processes* 14, 1909-1929, 2000

[168] Wolock, D.M.; McCabe Jr., G.J. Comparison of single and multiple flow direction algorithms for computing

[169] Wood, J. *The geomorphological characterisation of digital elevation models*. PhD Thesis. Department of Geography, University of Leicester. Leicester, UK. 1996.

Can be downloaded from *http://www.geog.le.ac.uk/jwo/research/dem_char/thesis/index.html*

Knowledge is of two kinds. We know a subject ourselves, or we know where we can find information on it.
SAMUEL JOHNSON